Inhaltsverzeichnis

1 Einleitung 5

2 Mikrocontroller und Robotik 9

3 AVR Mikrocontroller 13
- 3.1 Einführung . . . 14
- 3.2 Register . . . 17
- 3.3 Architektur . . . 27
- 3.4 Taktgeber . . . 31
- 3.5 Interrupts . . . 33
- 3.6 Digitale Inputs/Outputs . . . 35
- 3.7 Externe Interrupts . . . 37
- 3.8 Analog-zu-Digital Konverter . . . 39
- 3.9 Counters/Timers . . . 41
- 3.10 USART . . . 47

4 Elektronik 51
- 4.1 Ohm'sches Gesetz . . . 52
- 4.2 Berechnung des LED-Widerstands . . . 53
- 4.3 Spannungsteiler . . . 54

5 Robotic HomeLab Kit 57
- 5.1 Ganzheitliches Konzept der Robotik . . . 59
- 5.2 Robotik unterrichten . . . 61
- 5.3 HomeLab Hardware . . . 67
- 5.4 HomeLab Bibliothek . . . 71
  - 5.4.1 Bitweise Operationen . . . 73
  - 5.4.2 Pins . . . 76
  - 5.4.3 Analog-Digital-Wandler . . . 79
  - 5.4.4 Serielles Interface . . . 81

5.4.5 Timer . . . 85
5.4.6 Verzögerung . . . 94
5.4.7 7-Segment LED Display . . . 96
5.4.8 Alphanumerisches LCD . . . 97
5.4.9 Graphisches LCD . . . 99
5.4.10 Sensoren . . . 101
5.4.11 Motoren . . . 103

6 Praktische Beispiele 107
6.1 Anfang . . . 109
6.1.1 Installationsanweisung für Windows . . . 110
6.1.2 Die Softwareumgebung für Linux . . . 124
6.2 Digitale Input/Output Pins . . . 136
6.2.1 Light-emitting Diode (LED) . . . 137
6.2.2 Schalter . . . 143
6.2.3 Vermeidung von Kontaktprellung . . . 146
6.3 Timer und Verzögerungen . . . 153
6.3.1 Softwareverzögerungen . . . 154
6.3.2 Hardwareverzögerung . . . 157
6.3.3 Periodische Interrupts . . . 160
6.3.4 Aufgaben . . . 163
6.4 Anzeigen und Displays . . . 165
6.4.1 7-Segment-LED-Anzeige . . . 166
6.4.2 Alphanumerisches LCD . . . 170
6.4.3 Graphisches LCD . . . 173
6.4.4 Aufgaben . . . 175
6.5 Sensoren . . . 177
6.5.1 Potentiometer . . . 178
6.5.2 Thermistor . . . 182
6.5.3 Fotowiderstand . . . 186
6.5.4 Infrarot-Entfernungsmesser . . . 192
6.5.5 Ultraschall-Entfernungsmesser . . . 198
6.5.6 Übungen . . . 203
6.6 Motoren . . . 205
6.6.1 Gleichstrommotor . . . 205
6.6.2 Servomotor . . . 210
6.6.3 Schrittmotor . . . 215

6.6.4 Aufgaben . . . 219
6.6.5 Fragen . . . 221
6.7 Datenschnittstellen . . . 222
6.7.1 RS-232 . . . 223
6.7.2 ZigBee . . . 227
6.7.3 Aufgabe . . . 227
6.7.4 Fragen . . . 228

7 Beispielprojekt **229**
7.1 Bericht . . . 229
7.2 Mobile Roboterplattform . . . 231

8 Ausblick **247**

1 Einleitung

Dieses Buch wurde als praktisches Handbuch für Mikorcontroller und Robotik konzipiert, in der Hoffnung, dem Leser bei der Entwicklung von „smarten“Lösungen zu helfen. Darüber hinaus soll es auch dazu beitragen, die Begeisterung für den Ingenieurberuf bei jungen Menschen zu wecken. In erster Linie richtet sich das Buch an Schulen sowie Universitäten, aber auch an Bastler und Unternehmen, die im Rahmen von Projekten mit AVR Mikrocontrollern arbeiten möchten.

Ziel ist es, sowohl Lehrende als auch Lernende dabei zu unterstützen, rasch Ergebnisse zu erzielen: so soll Schülern und Studenten neues Wissen vermittelt und das Leben der Lehrenden erleichtert werden.

Plattform

Für die praktischen Beispiele in diesem Buch wird ein AVR ATmega128 Mikrocontrollersystem,das sogenannte HomeLab Kit, als grundlegende Plattform genutzt. Die AVR Mikrocontroller sind, neben Microchip PIC, die am weitesten verbreiteten in der Hobby-Robotik-Community. Darüber hinaus sind siebesonders gut für Lehrzwecke geeignet.

Das Buch ist systematisch in fünf Kapitel unterteilt:

1. Das erste Kapitel gibt eine kurze Einführung in die grundlegenden elektronischen Prinzipien und Berechnungen. Die vorgestellten Beispiele und Formeln sind sehr nützlich für viele folgende Aufgaben. Außerdem wird die Programmiersprache „C“kurz vorgestellt und ein paar grundlegende Anwendungsbeispiele werden diskutiert.

2. Im zweiten Kapitel wird am Beispiel des ATmega 128 ein Überblick über 8-bit AVR Mikrocontroller gegeben. Dieses Kapitel basiert weitgehend auf dem Datenblatt des ATmega Mikrocontrollers, stellt die benötigten Informationen jedoch

vereinfacht und übersichtlicher dar. Hier werden allgemein die Grundfunktionen eines Mikrocontrollers erörtert.

3. Dieses Kapitel führt die Hardware- und Softwareplattform ein, welche für Lehrzwecke entwickelt wurde und als Grundgerüst für weitere Beispiele in den folgenden Kapiteln dient. Ferner wird an dieser Stelle auch die „HomeLab Library“, eine für das Kit entwickelte Softwarebibliothek, welche vereinfachte Methodenaufrufe für häufig genutzte Funktionalitäten bietet, vorgestellt. Dies bietet dem Benutzer den Vorteil, dass er sich auf die Logik seiner Software konzentrieren kann, anstatt den Fokus auf das Programmieren von Registern legen zu müssen. Dabei berücksichtigt dieses Buch sowohl Windows- als auch Linux-Betriebssysteme.

4. Im vierten Kapitel werden schließlich praktische Beispiele sowie Übungen vorgestellt. Die Beispiele sind unterteilt in „Labs“(Übungsabschnitte) und decken die meisten elektro-mechanischen Bauteile und Funktionen ab. Jeder Abschnitt enthält zunächst eine kurze Beschreibung des theoretischen Hintergrunds sowie praktische Beispiele. Durch einfaches Kopieren der Beispielprogramme in den Controller hat der Nutzer direkt die Möglichkeit, die Funktionen zu nutzen. Am Ende jedes Unterkapitels finden sich Beispielaufgaben sowie Verständnisfragen zum jeweiligen Übungsabschnitt. Sämtliche Übungen sind nach ihrem Schwierigkeitsgrad gegliedert, damit sowohl Anfänger als auch Fortgeschrittene die für sie geeigneten Übungsaufgaben auswählen können. Darüber hinaus werden zum besseren Verständnis der Aufgaben Bezüge zu den betreffenden theoretischen Abschnitten dargestellt.

5. Im fünften Kapitel finden sich Tipps für die gemeinsame Arbeit an Problemstellungen. Hier wird illustriert wie man gemeinsam entwickelte Konzepte implementiert sowie präsentiert und wie Berichte verfasst werden. Dabei sind jene Themen, welche in einem Teambericht enthalten sein sollen, besonders hervorgehoben. Weiterhin wird an dieser Stelle auch ein Beispielprojekt vorgestellt, welches als Muster zum Erstellen eines präzisen Berichts für eigene Projekte dienen soll.

Wir hoffen, dass das Buch eine gute Hilfe sowohl für neue als auch für erfahrene Mikrocontroller- und Robotik-Enthusiasten darstellt und vielen Menschen diese Technologie näher bringt.

Bochum, im Sommer 2011

Sven Seiler

2 Mikrocontroller und Robotik

Mikrocontroller

Abbildung 2.1: The first microcontroller in the world: Intel 8048

Mikrocontroller sind im Wesentlichen Computer, welche auf einen einzelnen Chip mit integriertem Schaltkreis positioniert werden und aus einer Speichereinheit, einem Prozessor sowie Input-Output Schnittstellen bestehen. Sie werden für eine bestimmte Aufgabe programmiert, was bedeutet, dass sobald eine Änderung oder Verbesserung der Aufgabe nötig ist, ein neues Programm auf dem Chip installiert werden muss. Eigenschaften, welche den Mikrocontroller von anderen Computern (PC, Laptop, Server, etc.) unterscheiden sind:

Sämtliche Funktionen sind auf einem einzelnen, kleineren und kompakteren Chip gespeichert.

Er wird zur Durchführung einer bestimmte Aufgabe programmiert, um die Funktionalität zu ändern muss neue Software installiert werden.

Er verbraucht weniger Strom, da alle physischen Komponenten kleiner und energiesparender sind als die eines PC, Laptops oder Servers. Entwickler von Mikrocontrollern fokussieren einen geringen Energieverbrauch, sodass mobile Anwendungen länger betrieben werden können.

In- und Outputs, die auf einen bestimmten Zweck ausgelegt sind. Mikrocontroller besitzen so genannte Peripherie-Schnittstellen, welche Verbindungen zwischen mehreren Mikrocontrollern oder zwischen einem Mikrocontroller und einem Computer (z. B. USB, CAN, UART) ermöglichen sowie dabei helfen, reale physikalische Prozesse nachzuvollziehen (z. B. Schaltvorgänge, Temperaturmessungen, etc.) und nicht zuletzt auch die Steuerung von Umgebungen unterstützen (z. B. Motoren steuern, Alarme auslösen, etc.).

Mikrocontroller finden sich in diversen Gegenständen des täglichen Gebrauchs: in Haushaltsgeräten (z. B. Mikrowelle, Fernsehgerät), Spielzeugen (Lego NXT, sprechende Puppen), Beförderungsmitteln (Auto, Aufzug), etc. Der umfassende Einsatz von Mikrocontrollern ist möglich, da sie leicht zu programmieren sind und eine Vielzahl von Funktionen besitzen; folglich können sehr leicht neue Funktionen hinzugefügt sowie das Anwendungsniveau erhöht werden.

Robotik

Robotik ist die Wissenschaft, welche die Technologie und das notwendige Wissen zum Bau von Robotern miteinander kombiniert. Aufgrund des rasanten technologischen Fortschritts ist der Begriff „Roboter“, der als automatisierte Maschine den Menschen ersetzt, nicht mehr klar definiert. Roboter sind nicht länger nur humanoide Roboter, Roboterhände in Fertigunggstraßen, Autopiloten in Flugzeugen, aus lebenden Neuronen bestehende künstliche Intelligenz oder einfache Reinigungsroboter; der Begriff „Roboter“bezeichnet vielmehr auch Computersoftware, welche für den Menschen gedachte Aufgaben ausführt (z. B. Berichte erstellen). Es ist allseits bekannt, dass Roboter gebaut werden, um den Menschen bei bestimmten Aufgaben zu ersetzen. Eine Vielzahl von Gründen rechtfertigt dieses: gefährliche Arbeitsbedingungen, günstigere Produktion, monotone Arbeit bei der Menschen zu Fehlern neigen, neue Systeme, die so komplex und zeitkritisch sind, dass automatisierte Maschinen diese besser erledigen können als Menschen.

Mikrocontroller in der Robotik

Aufgrund der Weitläufigkeit der Robotik, konzentrieren wir uns in diesem Buch auf die Hobby-Robotik. Diese Systeme sind nicht allzu komplex und es ist möglich sie eigenständig zu implementieren. Häufig genutzte Mikrocontroller in der Robotik sind:

Atmel AVR Mikrocontroller (ATmega, ATtiny, etc.)

Microchip Technology PIC Mikrocontroller (PIC16, PIC24, etc.)

Mikrocontroller, die auf der ARM-Technologie basieren.

Drittanbieter haben sehr häufig Entwicklungsplatinen und Umgebungen gebaut, die auf den zuvor genannten Mikrocontrollern basieren. Zum Beispiel: Arduino (VAR), BASICStamp (PIC) und Lego NXT (ARM). Die zur Entwicklung von HomeLab notwendigen Grundvoraussetzungen, welche in diesem Buch beschrieben werden, basieren auf dem AVR ATmega127 Mikrocontroller.Aus der Vielzahl der zur Verfügung stehenden Mikrocontroller und Entwicklungsplatinen gilt es nun, das für den jeweiligen Bedarf am besten geeignete Produkt auszuwählen. Allgemein lassen sich die folgenden 4 Eigenschaften unterscheiden: Preis, technische Merkmale, Entwicklungsumgebung und Kundenbetreuung. Bemerkenswerte technische Merkmale sind:

Arbeitsgeschwindigkeit des Prozessors - legt die Arbeitsgeschwindigtkeit des Chips fest

Speicherkapazität des Programms - bestimmt die Größe des Programms, das auf dem Chip installiert werden kann

Datenspeicherkapazität - gibt die mögliche Datenverarbeitungsmenge des Programms an

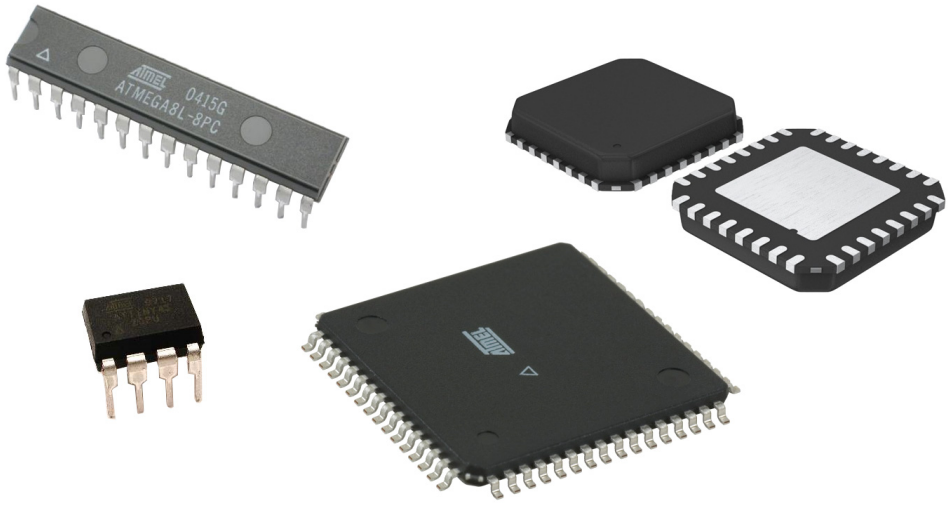
Anzahl der Input- / Output-Schnittstellen sowie deren Funktionen - unterschiedliche Schnittstellen bieten differenzierte Möglichkeiten

Anzahl der Timer - wichtig für das Zeitverhalten der Anwendung

Energieverbrauch - von großer Bedeutung für mobile Anwendungen

In diesem Buch verwenden wir PC-Software als Entwicklungsumgebung, mit der Programme erstellt und kompiliert und auf die Mikrocontroller geladen werden können sowie Zugriff auf die laufenden Programme hergestellt werden kann, um so mögliche Fehler zu entdecken. Da diese Software während der Entwicklungsphase des Programms als primäre Entwicklungsumgebung dient, wird die einfache und angenehme Arbeitsweise während der Nutzung deutlich. Dies führt schließlich zur vierten Eigenschaft, der Kundenbetreuung. Hier ist es wichtig, dass auf Hilfe und Unterstützung bezüglich diverser Problemstellungen so einfach wie möglich zugegriffen werden kann. Unter Berücksichtigung der vier genannten Merkmale, sollte es schließlich möglich sein, die geeignete Entwicklungsplatine zu finden.

3 AVR Mikrocontroller



Das folgende Kapitel stellt den AVR Mikrocontroller vor, auf welchem dieses Buch basiert. Trotz seiner geringen Größe besitzt der Mikrocontroller eine Vielzahl von Funktionen, welche von Atmel in einem fast 400 Seiten umfassenden Handbuch dokumentiert wurden. Darüber hinaus gibt es noch einige weitere Dokumente, die sich mit der Thematik befassen. Sämtliche Informationen werden in dem folgenden Abschnitt in komprimierter Form dargestellt, um dem Leser so einen schnellen und einfachen Überblick über das Thema zu geben. Zugleich dient diese Übersicht dazu, ungeübten Nutzern das Verständnis des AVR Mikrocontrollers näher zu bringen und zeigt ihnen, wie das Datenblatt zu lesen ist.

3.1 Einführung

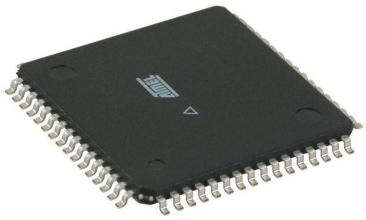


Abbildung 3.1: ATmega128 in SMT package (TQFP64, to be precise)

AVR ist eine Serie aus 8-bit RISC Microcontroller produziert von Atmel. AVR folgt der Harcard-Architektur und hat daher separate Programm- und Datenspeicher. Für das Programm hat er einen internen überschreibbaren Flash-Speicher, für statische Daten (SRAM) und EEPROM-Speicher. Die Taktfrequenz des Controllers liegt normalerweise bei 16MHz und schafft fast 1 MIPS pro 1MHz-Zyklus.

Die Produktion des AVR Microcontroller begann 1997 und heute ist AVR einer der am beliebtesten Controller der freischaffenden Elektronik-Ingenieure. Dank der günstigen Entwicklungstools, der diversen Peripherals in einem Paket und des niedrigen Energiebedarfs kam der anfängliche Erfolg. Heute gibt es noch einen anderen Grund sich für den AVR zu entscheiden: die Masse an Information und an Tutorials die sich über die Jahre angesammelt haben. Die AVR Technologie altert natürlich, aber um konkurrenzfähig zu bleiben stellt Atmel neue AVR Microcontroller her, mit up-to-date peripherals und 16- und 32-bit Bus, Erste aus der 8-bit kompatiblem XMega Serie und Letztere aus der neuen AVR32 Serie.

Basierend auf den Typ der Anwendung, gibt es unterschiedliche Typen von AVR Microcontroller, jeder mit einer unterschiedlichen Konfiguration. Die meisten der AVRs gehören zur megaAVR Serie, sie haben einen großen Programm-Speicher. Als Gegenstück gibt es die tinyAVR Serie, mit weniger Funktionen. Zusätzlich gibt es noch unterschiedliche Microcontroller Serien speziell zum Kontrollieren von USB, CAN, LCD, ZigBee, automatics, Beleuchtung und batteriebetriebene Geräte.

Der folgende Text beschreibt die Hauptfeatures der megaAVR Serie Microcontroller, am Beispiel des beliebtesten Controller dieser Serie, des ATmega128. Dieser Controller ist auch im HomeLab-Kit enthalten. Generell alle AVR Microcontroller Registernamen, Bedeutungen und Benutzungen sind in einer Weise geregelt , welche es möglich macht Beispiele auch mit anderen Controller zu nutzen, in dem man nur kleine Änderungen durchführt.Der Hautunterschied liegt in den Peripherals.Die Codebeispiele dieser Einführung sind mit AVR LibC in Assembler und C geschrieben .

äußerer Aufbau

Abbildung 3.2: ATmega32 in 40-pin DIP casing

Wie alle anderen Controller ist der AVR in einer Standarthülle gepackt. Das traditionelle Gehäuse ist DIP (bzw. DIL). DIP ist eine so genannte Casing-On-Legs. Alle Pins treten wie Beine , ca 5mm in Länge, aus dem schwarzen Plastikgehäuse hervor.DIP Gehäuse sind eine gute Wahl für Hobbyanwendungen und Prototypen, weil es dafür günstige Sockets gibt. Daher kann der Microcontroller einfach ersetzt werden sollte er Ausfallen. Die Beine sind aber auch der Nachteil des DIP Gehäuses, da man dafür Löcher in die Platine bohren muss.

Die Surface-Mount-Casings (SMT, bzw. SMD) sind viel kompakter, weil die Pins dafür gedacht sind direkt auf die Platine gelötet zu werden ohne zu Bohren. AMT Microchips sind dünne münzen-große rechteckige Gehäuse mit Pins von ca. 1mm Länge. Eine ruhigere Hand und präzisere Werkzeuge werden zum löten von SMT Chips benötigt.

AVRs gibt es als DIP und SMT Gehäuse. Das Layout der Pins ist vom Design her so logisch und elektronisch ausgeglichen wie es nur möglich ist. Zum Beispiel

sind bei größeren Chips die Ground- und Supply-Pins auf mehreren Seiten des Microcontrollers, die Pins für einen Externen Oszillator sind nahe des Ground-Pins, die Bus-Pins sind durchnummeriert, die Communication-Pins sind nebeneinander etc. AVRs digitale Pins sind kompatibel mit TTL/CMOS Voltwerten. Bei 5V Supply-Spannung, 0 bis 1 V sind Logische Null (Zero), auch Zero, Null, 0 Low, Ground oder GND genannt. Bei Gleicher Supply-Spannung, 2 bis 5.5V sind die logische Eins, auch one, 1, high, genannt. Dieses Typ von weiten Spannungsunterschieden gibt es nur im Input, die Outputspannung an einem Pin mit keinem Load ist immer noch 0 V oder nahe der Supply-Spannung, je nach Status des Pins.Die erlaubte Analogspannung an den ADC Kanälen ist 0-5.5V.

ATmega128 Um die folgenden Beispiele am ATmega 128 besser zu verstehen, gibt es am Ende des Textes ein Pinout-Schema des ATmega128 (SMT Package).An jedem Pin ist ein Text mit der Nummer, primären Funktion und sekundären (alternativen) Funktion in Klammen. Supply Pins sind GND und VCC.AVCC und AREG sind die analog zu digital Konverter Supply- und ReferenceVoltage-Pins.XTAL1 und XTAL2 sind für den Anschluss einen externen Schwingquarzes, Resonator oder Taktgebers. Die Pins PB0 bis PG4 markieren die Bits der Input-Output Buses. Die sekundären Funktionen werden in dem entsprechenden Kapitel besprochen.

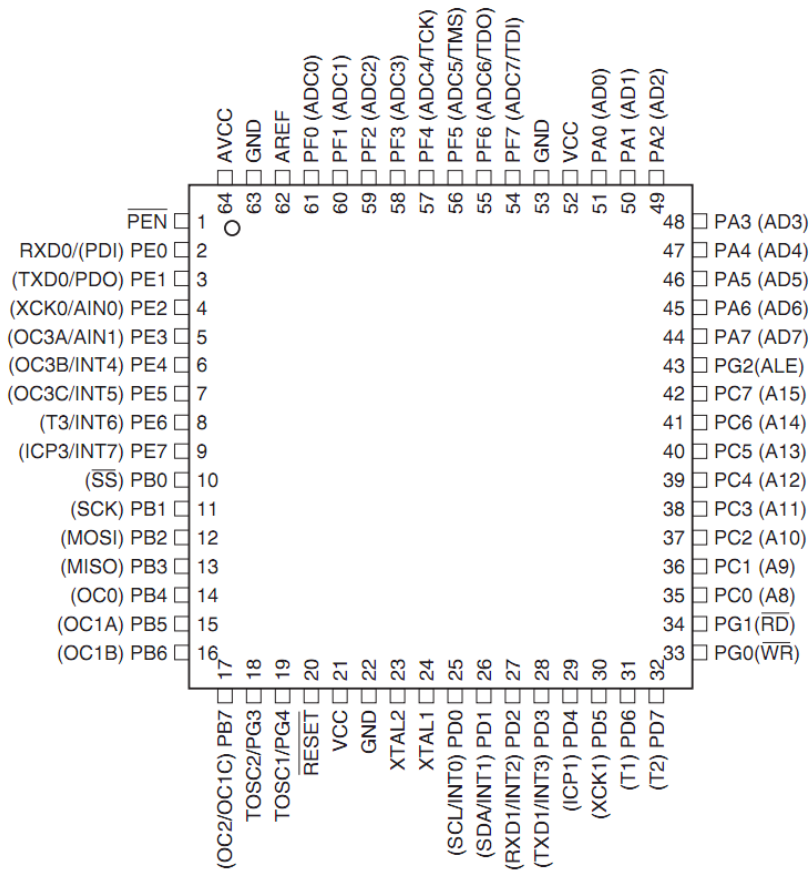


Abbildung 3.3: ATmega128 pinout

3.2 Register

Typischerweise ist das Register eines der Bestandteile eines Mikrocontrollers, die ein Anfänger nur mühsam versteht. Im Umgang mit Mikrocontrollern ist es jedoch unumgänglich diese Komponente zu kennen. Auch für die weitere Arbeit mit diesem Buch ist der Leser dazu angehalten, sich das Konzept des Registers anzueignen. Daher wird es im nun folgenden Text auf eine möglichst verständliche Weise erklärt, sodass auch ein Anfänger ein Gespür für das Register bekommt.

Essence

Abbildung 3.4: Tasten eines Kassettenrekorders

Ein Register ähnelt einem Panel von Tasten eines Haushaltsgerätes. Es verfügt über Schalter, die ein- und ausgeschaltet werden können. Das beste Beispiel hierfür ist ein Kassettenrekorder. Zur Erinnerung, ein Kassettenrekorder verfügt über 6 Tasten von links nach rechts:

Aufnahme

Zurückspulen

Abspielen

Vorspulen

Stopp

Pause

Jede Taste führt eine Funktion aus, jedoch nur wenn sie auch korrekt verwendet wird. So macht die Stopp-Taste solange nichts, bis die Kassette abgespielt wird. Erst dann kann die Funktion ausgeführt werden und die Wiedergabe wird gestoppt. Dagegen können Vorlauf- und Rückspultasten zu jeder Zeit betätigt werden, denn die Kassette kann in beide Richtungen gespult werden, unabhängig davon ob sie gerade abgespielt wird oder nicht. Die Aufnahmetaste startet nur dann eine Aufnahme, wenn sowohl die Abspielen- als auch die Aufnahmetaste zeitgleich gedrückt werden. Eventuell hat mancher einmal versucht, mehrere oder alle Tasten auf einmal zu betätigen. Dieses hat vermutlich eine unerwartete Funktion des Kassettenrekorders ausgelöst oder ihn sogar beschädigt.

Die Register von Mikrocontrollern verhalten sich wie Tasten eines Kassettenrekorders - jede Taste führt eine Funktion aus, wenn sie korrekt verwendet wird. Werden die falschen Tasten betätigt, wird der Mikrocontroller in der Regel nicht direkt beschädigt, aber er wird nicht funktionieren. Tatsächlich gibt es keine Tasten im Register sondern stattdessen eine Vielzahl von Transistoren, welche die Stromversorgung ein- und ausschalten. Einfachere Mikrocontroller haben 8 transistorbasierte Schalter in einem einzigen Register. Unter einem Register kann man sich eine 8-Bit Zahlenfolge vorstellen, wobei jedes Bit durch den Status eines dieser Schalter gekennzeichnet wird. Zum Beispiel kann ein Bit-Wert von 1 bedeuten, dass der Schalter an ist, ein Wert von 0 hieße dann, dass er aus ist.

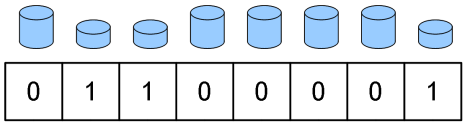


Abbildung 3.5: „Tasten“des Registers und Bitwerte

Da der Status eines Registerschalters einfach durch eine Nummer abgebildet werden kann, lässt sich ein Register mit einem Speicher vergleichen, der Daten in Form genau einer Zahl speichern kann. Durch diesen Vergleich wird deutlich, dass Register tatsächlich Speicherslots sind. Der Unterschied zwischen einem Register und einem Speicherslot besteht darin, dass letzterer lediglich Informationen speichert, während die Informationen in einem Register tatsächlich etwas steuern. So sorgt zum Beispiel der Binärwert 01100001 in einem Register dafür, dass drei imaginäre Tasten betätigt werden wodurch eine Funktion ausgelöst wird.

Während es bei einem Kassettenrekorder möglich ist, jeden Knopf einzeln zu betätigen, ergeben sich bei einem Register Schwierigkeiten, wenn nur der Wert eines Schalters oder Bits geändert werden soll. Hier ist es gewöhnlich notwendig, den gesamten Inhalt der Registers zu ändern. Bevor jedoch detaillierter auf die Thematik der Bit-Veränderung eingegangen wird, muss zunächst klargestellt werden, dass ein Mikrocontroller mehrere Register enthält. Die Vielzahl der Register führt dazu, dass diese voneinander unterscheidbar gemacht werden müssen, was durch Kennzeichnung mit einem Namen geschieht, wie zum Beispiel „PORTB“. Hierdurch wird dem Entwickler die Arbeit erleichtert, da jedem Namen auch eine eigene numerische Adresse zugewiesen wird.

Gebrauch

Zur Programmierung eines Registers oder auch dazu, um Werte aus diesem abzulesen muss er als Variable in C deklariert werden. Das folgende Beispiel veranschaulicht, wie ein Binärwert einem imaginären Register REG zugewiesen wird und dann in der Variable *reg* abgelegt wird. Binärwerte werden an einem vorangestellten 0b (leading zero) erkannt, sodass der Compiler das binäre Zahlensystem erkennt.

```
REG = 0b01100001;
unsigned char reg = REG;
```

Die Schwierigkeit liegt nicht darin, Registerwerte zu schreiben und auszulesen, sondern vielmehr darin, ein einzelnes Bit zu verändern. Hierzu sind Kenntnisse der Binärmathematik sowie im Gebrauch diverser Zahlensysteme notwendig. Es ist auch möglich, nur mit Binärzahlen zu arbeiten. Da diese aufgrund ihrer Länge die Programmierarbeit jedoch erschweren können, verwenden die meisten Programmierer die kürzeren Hexadezimalzahlen.

0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111

8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

Abbildung 3.6: Hexadezimalzahlen

Im Hexadezimalsystem gibt es nicht nur die Ziffern 0 und 1 wie im binären System oder 0 bis 9 gemäß dem Dezimalsystem, sondern stattdessen 0 bis F. Eine Hexadezimalzahl besteht aus vier Bits. Die Abbildung auf der rechten Seite veranschaulicht die Binärzahlen und deren zugehörige Hexadezimalwerte. Binärzahlen werden in Hexadezimalzahlen umgewandelt indem vier Bits zugleich, beginnend bei dem niedrigsten, gelesen werden. Die Bitfolge wird von links nach rechts gelesen und startet bei 0. So ist zum Beispiel das niederwertigste Bit (an Stelle 0) 0 und das mit dem höchsten Wert (an Stelle 3) 1. Im vorangehenden

Beispiel ist der Binärwert der Registers 01100001, der im Hexadezimalsystem 61 entspricht und in C als 0x61 (leading zero) geschrieben wird.

Um einzelne Bits innerhalb einer Zahl zu verändern (Register, Variable oder ähnliches) müssen Binäroperationen angewandt werden. Binäroperationen sind Vorgänge zwischen zwei Binärzahlen, wobei jedes Bit der Zahlen durch eine eigene logische Operation abgebildet wird. Mikrocontroller unterstützen generell vier unterschiedliche Binäroperationen. Der folgende Abschnitt beschreibt die logische Operation hinter diesen vier Binäroperationen mit einzelnen sowie mehreren Bits.

A	¬
0	1
1	0

A	B	∧
0	0	0
0	1	0
1	0	0
1	1	1

A	B	∨
0	0	0
0	1	1
1	0	1
1	1	1

A	B	⊕
0	0	0
0	1	1
1	0	1
1	1	0

Abbildung 3.7: Negation, logische Multiplikation, logische Addition und exklusive Disjunktion

Negation / Inversion
Eine Negation kehrt den Wert eines Bits in das jeweilige Gegenteil um. So wird aus einer 0 eine 1 und umgekehrt. In C wird eine Negation durch “~“vorgenommen.

Logische Multiplikation / Konjunktion
Bei der Multiplikation zweier Bits ist das Ergebnis 1 für den Fall, dass beide Bits den Wert 1 besitzen, ansonsten 0. Eine Logische Multiplikation wird in C durch “&“dargestellt.

Logische Addition / Disjunktion
Eine Addition von zwei Bits ergibt 1 wenn zumindest eines der Bits den Wert 1 hat und 0 wenn beide Bits den Wert 0 haben. In C kennzeichnet „|“die Logische Addition.

Exklusive Disjunktion / Exklusives ODER / XOR
Das Ergebnis einer Exklusiven ODER Operation ist 1 bei zwei unterschiedlichen Bits (also wenn ein Bit den Wert 1 besitzt, das andere den Wert 0), ansonsten 0. Eine Exklusive Disjunktion wird in C durch „^ “dargestellt.

Man benötigt nicht mehr als diese vier Operationen, um einzelne Bits zu ändern. Da die Theorie allein vermutlich nicht ausreichen mag, enthalten die folgenden Abschnitte einige Anwendungsbeispiele.

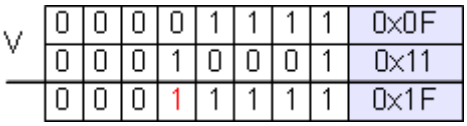


Abbildung 3.8: Ein einzelnes Bit „high“setzen

Ein einzelnes Bit “high“setzen Um einzelne oder mehrere Bits in einem Register “high“zu setzen (1) muss eine logische Addition durchgeführt werden. Hierfür muss ein Operand das Register sein und ein anderer die Binärzahl, wobei das einzige „high“Bit jenes ist, was ein einem Register „high“gesetzt werden muss. Diese Binärzahl wird Bitmaske genannt. Nachfolgend ist der C-Code für diese Operation abgebildet:

```
// Annahme REG = 0x0F
REG = REG | 0x11; // Erste Methode
REG |= 0x11;      // Zweite Methode
// Ergebnis REG = 0x1F
```

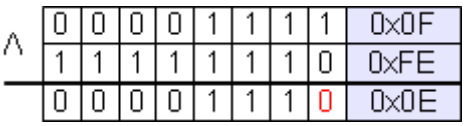


Abbildung 3.9: Ein einzelnes Bit “low“setzen

Ein einzelnes Bit “low“setzen Um einzelne oder mehrere Bits in einem Register “low“zu setzen (0) ist eine logische Multiplikation nötig. Dabei muss ein Operand das Register und ein weiterer eine Bitmaske sein, in welcher das einzige low-Bit jenes ist, welches im Register „low“gesetzt werden soll.

```
// Annahme REG = 0x0F
REG = REG & 0xFE; // Erste Methode
REG &= 0xFE;      // Zweite Methode
// Ergebnis REG = 0x0E
```

⊕									
	0	0	0	0	1	1	1	1	0x0F
	0	0	0	1	0	0	0	1	0x11
	0	0	0	1	1	1	1	0	0x1E

Abbildung 3.10: in einzelnes Bit invertieren

Ein einzelnes Bit invertieren / umkehren Zum Invertieren einzelner oder mehrerer Bits eines Registers muss eine Exklusive Disjunktion angewandt werden. Hierzu ist ein Operand das Register, der andere muss eine Bitmaske sein, in der das einzige high Bit jenes ist, welches invertiert werden soll. Der C-Code für diese Operation ist unten abgebildet:

```
// Annahme REG = 0x0F
REG = REG ^ 0x11; // Erste Methode
REG ^= 0x11;      // Zweite Methode (nur eine pro Inversion verwenden
  )
// Ergebnis REG = 0x1E
```

¬									
	0	0	0	0	1	1	1	1	0x0F
	1	1	1	1	0	0	0	0	0xF0

Abbildung 3.11: Invertieren aller Bits

Das gesamte Register invertieren Durch die Negationsoperation werden sämtliche Bits eines Register invertiert. Diese Operation besteht aus nur einen Operanden und ist daher unär. Der hierfür benötigte C-Code ist unten abgebildet:

```
// Annahme REG = 0x0F
REG = ~REG;
// Ergebnis REG = 0xF0
```

∧									
	0	0	0	0	1	1	1	1	0x0F
	0	0	0	0	0	0	0	1	0x01
	0	0	0	0	0	0	0	1	0x01

Abbildung 3.12: Den Wert eines Bits auslesen

Den Wert eines einzelnen Bits auslesen Sollen ein oder mehrere Bits aus einem Register gelesen werden muss ebenfalls die logische Multiplikation angewandt werden. Einer der Operanden ist das Register, der zweite eine Bitmaske, in welcher das einzige high-Bit jenes ist, welches aus dem Register gelesen werden soll. Unten ist der C-Code für diesen Vorgang dargestellt.

```
// Annahme REG = 0x0F
unsigned char x = REG & 0x01;
// Ergebnis x = 0x01
```

Ein Bit verschieben Viele Programmiersprachen verfügen mittlerweile über einige zusätzliche Bitoperationen, was die Programmierarbeit erleichtert. Hierzu gehören die Operationen, welche die Bits einer Binärzahl nach rechts oder links verschieben. Im Umgang mit Registern liegt der größte Vorteil dieser Operationen darin, dass Bitfolgen zu Bitmasken konvertiert werden können und umgekehrt.

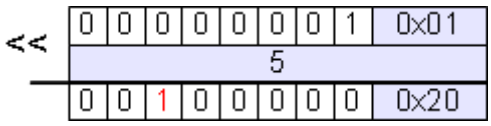


Abbildung 3.13: Linksshift

Das Bild auf der rechten Seite zeigt einen Linksshift. Auch wenn ein Bitshift keine logische Operation darstellt und kein zugehöriges Symbol hat, wird diese Operation in C mit "«"gekennzeichnet. Eine Linksshiftoperation wird genutzt, um eine Bitfolge in eine Bitmaske umzuwandeln. Um beispielsweise die Maske für das sechste Bit (Rang 5) zu erhalten, muss das erste Bit um fünf Stellen nach links verschoben werden. In C wird diese Operation wie folgt geschrieben:

```
REG = 0x01 << 5;
// Ergebnis REG = 0x20
```

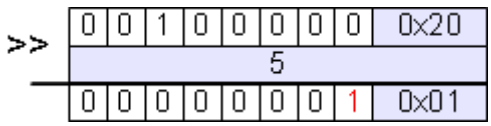


Abbildung 3.14: Rechtsshift

Der Rechtsshift funktioniert auf die gleiche Weise und wird in C durch "»"dargestellt. Rechtsshiftoperationen dienen dazu, den logischen Wert eines Bits aus der Bitmaske zu erhalten. An einem vorherigen Beispiel wurde bereits gezeigt, wie der Wert eines einzelnen Bits gelesen werden konnte. Ist dieses Bit nun nicht das niederwertigste, sondern zum Beispiel eines mit dem Rang 5, wäre das Ergebnis entweder 0x20 oder 0x00. Allerdings wird manchmal ein Ergebnis von 1 oder 0 benötigt und genau hier kommt die Rechtsshiftoperation zur Anwendung. Der Rechtsshift aus dem Beispiel rechts sieht in C folgendermaßen aus:

```
// Annahme REG = 0x20
unsigned char x = REG >> 5;
// Ergebnis x = 0x01 (or simply 1)
```

Wird durch diese Operationen ein Bit vom niedrigsten Rang nach rechts oder vom höchsten Rang nach links verschoben, verschwindet es. Einige Programmiersprachen verfügen jedoch über rotierende Bitshiftoperationen, bei denen ein Bit nicht verschwindet, sondern vom niedrigsten zum höchsten Rang wandert oder umgekehrt. In C gibt es diese Operationen nicht, sie können jedoch bei Bedarf vom Programmierer selbst geschrieben werden.

Sämtliche Bitoperationen sind nicht nur mit Registern durchführbar, sondern auch mit Variablen und Konstanten. Letztere können jedoch ausschließlich als Operanden, nicht als Ergebnis genutzt werden.

AVR Register

Um tatsächlich mit dem Register eines Mikrocontrollers zu arbeiten, ist es notwendig zu wissen, wie dieser spezielle Mikrocontroller verwendet wird. Zu jedem Mikrocontroller gibt es ein oder mehrere Datenblätter, welche seine Struktur und Funktionalität beschreiben. Dieses Datenblatt enthält auch Informationen über die Register. Der folgende Abschnitt dient dazu, die Registerbeschreibungen der AVR Datenblätter zu verstehen.

Bit	7	6	5	4	3	2	1	0	
	RXCn	TXCn	UDREn	FEn	DORn	UPEn	U2Xn	MPCMn	UCSRnA
Read/Write	R	R/W	R	R	R	R	R/W	R/W	
Initial Value	0	0	1	0	0	0	0	0	

Abbildung 3.15: Eines der AVR Register, Datenblattansicht

Die Abbildung zeigt das UCSRnA Register des ATmega128. UCSRnA bezeichnet „USART Control and Status Register A“. Dieses Register wird verwendet um das USART Modul der AVR Mikrocontroller zu konfigurieren und deren Status auszulesen. Sämtliche AVR Registernamen werden in Großbuchstaben geschrieben, jedoch enthält der Name auch ein kleines n, welches den Index des Moduls kennzeichnet. Der ATmega128 verfügt über zwei nahezu identische USCART Module, die jedoch nicht getrennt beschrieben werden, sodass der Nutzer das n als 0 oder 1 lesen muss. Aus diesem Grund hat der ATmega128 die Register USCR0A und UCSR1A.

Der Inhalt des Registers wird durch eine Box mit 8 Feldern und einem dicken schwarzen Rahmen dargestellt. Jedes Feld steht hier für ein Bit. Die Biträngе sind oberhalb der Box abgebildet - von links nach rechts ansteigend. Da der AVR ein 8-Bit Mikrocontroller ist, haben auch die meisten seiner Register 8-Bit. Es gibt auch Ausnahmen, so sind einige Register 16-Bit groß, aber tatsächlich bestehen sie aus zwei 8-Bit Registern. So wie jedes Register einen Namen hat, hat auch jedes Bit innerhalb des Registers einen Namen - so wie die Tasten an einem Kassettenrekorder. Jedes Bit wird im Datenblatt beschrieben. Wie Registernamen sind Bitnamen ebenfalls Abkürzungen und das kleine n entspricht jeweils dem Modulindex. Einige Register nutzen weniger als 8 Bit, in diesem Fall ist das Feld eines solchen Bits mit einem Trennstrich markiert.

Unterhalb der Bits des Registers finden sich im Datenblatt zwei Zeilen. Die erste gibt an, ob das Bit lesbar (R), beschreibbar (W) oder beides (R/W) ist. Kann beispielsweise der Status eines Bits nicht überschrieben werden, so bleibt das Bit unverändert, auch wenn das mit dem Programm versucht wurde. Durch Auslesen eines beschreibbaren Bits wird stets ein spezifischer Wert ausgegeben, der im Datenblatt angegeben ist. In der zweiten Zeile wird der Standardwert dargestellt, den das Bit nach dem Reset des Mikrocontrollers hat.

Während die AVR Registernamen auf eine tatsächliche Adresse im Speicher verweist, enthalten Bitnamen den Rang und das zugehörige Bit. Daher ist eine Transformation der Namen zu Bitmasken mit einer Shiftoperation notwendig, wenn man mit Bits in einem Register arbeiten will. Der folgende Code zeit einige Beispielzeilen für die Nutzung des USART 0 Modul Registers.

```
// Setze TXC0 Bit high
UCSR0A |= (1 << TXC0);

// Setze U2X0 Bit low
UCSR0A &= ~(1 << U2X0);

// Lies den Wert aus UDRE0 Bit(Maske)
unsigned char u = (UCSR0A & (1 << UDRE0));

// An dieser Stelle ist der Wert entweder 0 oder 32,
// wodurch die Nutzung in einer logischen Operation aktiviert wird
if (u)
{
   // Invertiere MPCM0 Bit
   UCSR0A ^= (1 << MPCM0);
}

// Manchmal ist es notwendig einen spezifischen 1 oder 0 Wert zu
  erhalten
// Dazu muss das Bit nach rechts verschoben werden
u >>= UDRE0;

// Nun ist der Wert entweder 0 oder 1
```

3.3 Architektur

Der AVR hat einen internen 8-Bit Daten-Bus, durch den Daten zwischen der arithmetisch-logischen Einheit (ALU), dem Status Register (SREG), dem Befehlszähler (PC), dem Random-Access-Memory (SRAM), und den Peripherieschnittstellen bewegt werden können. Das Programm, eine Reihe an Befehlen, die in der ALU ausgeführt werden, resultiert aus einer Adresse im Flash-Speicher, spezifiziert vom Befehlszähler. Der ALU hat 32 8-bit Mehrzweckregister, welche als Operanden genutzt werden, wenn Befehle ausgeführt werden.

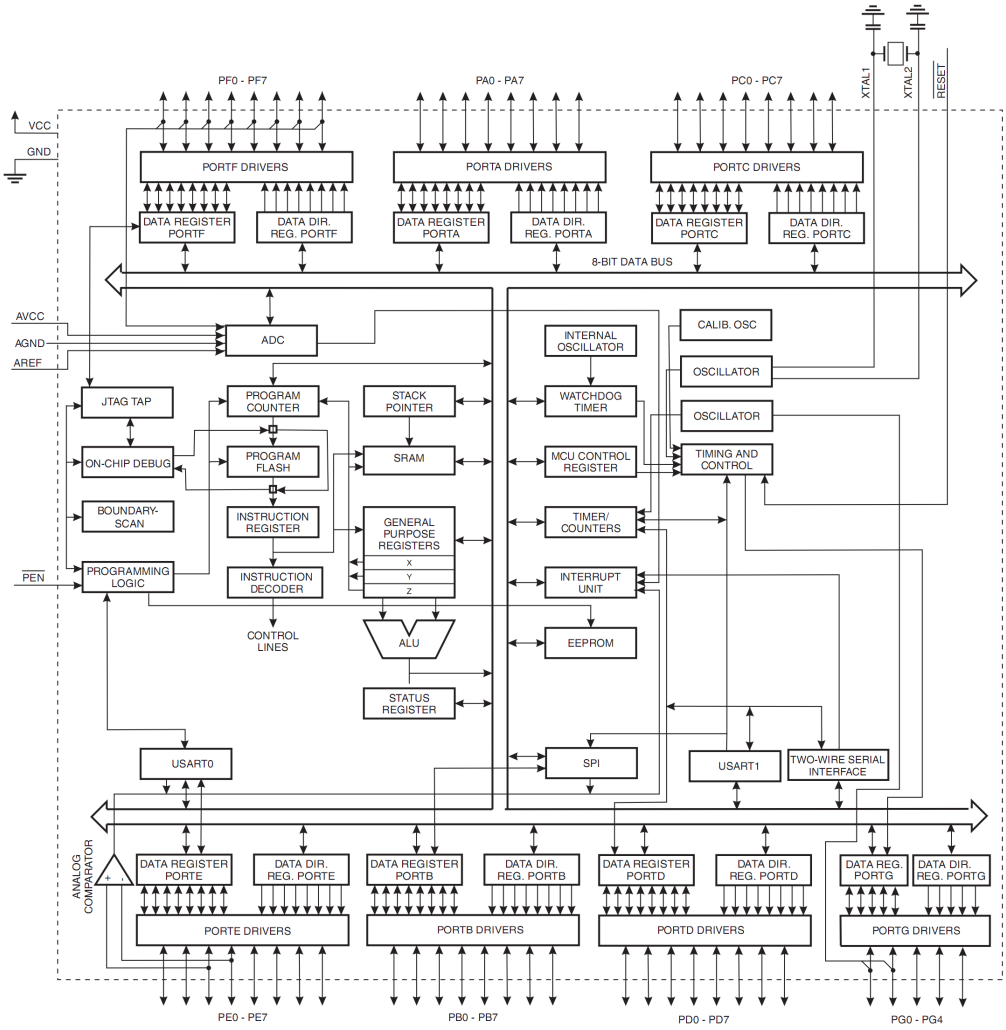


Abbildung 3.16: Block diagram of ATmega128

Befehls-Pipeline

Der AVR hat eine zweistufige Befehls-Pipeline. Während ein Befehl ausgeführt wird, wird der nächste aus dem Programmspeicher geholt. Darum sind beim Ausführen einer „Jump-Instruction“zwei Takte notwendig, um die Bedingungen des Jumps zu erfüllen. Neue Befehle werden immer von der nächsten Speicheradresse geholt. Dadurch ist es notwendig die vorherigen Befehle zu löschen und

neue zu laden, wenn man zu einer neuen Adresse springt, da der Ort an dem der Befehl geholt wurde nun nicht mehr stimmt.

Mehrzweckregister

Mehrzweckregister R0-R31 sind wie Buffer, um mit Speicher- und Peripheriedaten zu arbeiten und sie zu speichern. Sie vereinfachen die Architektur des Prozessors, weil sie schnellen Zugang der ALU erlauben und der Datenbus nicht immer benutzt werden muss um Operanden aus dem Speicher zu lesen. Mehrzweckregister werden für alle datenbezogenen arithmetischen und logischen Abläufe benutzt.

Wenn man in Assembler programmiert, ist es möglich die benötigten Daten im Mehrzweckregister zu speichern. Programmiert man in C und muss eine Variable im Mehrzweckregister speichern, wird die Variable zusätzlich als „Register“definiert. Zum Beispiel:

```
register char x;
```

Befehlssatz

Der Befehlssatz der meisten AVRs besteht aus 90-133 verschiedenen Befehlen. Der ATmega128 hat 133 Befehle. Befehle haben entweder ein, zwei, oder keine Operanden. Die meisten Befehle benötigen nur einen Takt zur Ausführung, komplexere Befehle können jedoch bis zu fünf Taktzyklen benötigen. Für den XMega, den Nachfolger des AVR, wurden viele Befehle modifiziert, damit weniger Taktzyklen genutzt werden. Die meisten Befehle im AVR werden benutzt für Jumps, um Daten zu bewegen und zu vergleichen und um arithmetische Berechnungen auszuführen. Ein Status-Register wird genutzt um Berechnungen und Vergleiche auszuführen. Es speichert den Output-Status der ALU - ob das Ergebnis negativ, positiv, null, den maximal erlaubten Wert (8 Bits) überschreitet, oder einen Bit in die nächste Operation transferieren muss, usw. (Es gibt noch ein paar kompliziertere Fälle).

Example

Die ist ein Teil eines in Assembler geschrieben Codes, der bloße Befehle beinhaltet, welcher eine 5 zum Byte in der RAM-Adresse $ 100 (dezimal 256) hinzufügt. Diese Befehle gibt es alle im AVR.

```
ldi   r1, 5       ; Lädt die Konstante 5 in das Mehrzweckregister r1
lds   r2, $100    ; Lädt das Byte aus dem Speicher in das Register r2
add   r2, r1      ; Fügt den Wert von r1 zu r2 hinzu
sts   $100, r2    ; Schreibt den Wert von r2 zurück in den Speicher
```

Programmstapelspeicher (stack)

Ein Stapelspeicher (stack) ist eine Datenstruktur, bei der der letzte in den Speicher geschriebene Wert, als erstes ausgelesen wird. Die Stacks im AVR können genutzt werden, wenn man mit Subroutinen, Interrupts und temporären Daten arbeitet. Bevor eine Subroutine oder ein Interrupt ausgeführt wird, speichert der Stack die Adresse im Befehlszähler, bei welcher das Programm unterbrochen wurde. Wenn die Subroutine oder der Interrupt ausgeführt wurde, wird die Adresse wieder aus dem Stack gelesen, und das Programm fährt von der Adresse vor der Unterbrechung wieder fort. Das Speichern temporärer Daten im Stack wird normalerweise dann genutzt, wenn man mit kleineren Stücken Code arbeitet, welche keinen reservierten Speicher während ihrer Ausführung benötigen. Einfache Assembler Programme werden normalerweise so geschrieben, dass es nicht notwendig ist, den Stack zu nutzen, aber wenn das Programm viele Variablen und Funktionen enthählt, verwendet der Compiler den Stack automatisch.

Der Stack der MegaAVR-Serie ist physikalisch im Random-Access-Memory zu finden. Einige Modelle der tinyAVR-Serie besitzen jedoch keinen Random-Access-Memory sodass der Stack hier durch eine kleine, separate und begrenzte Speichereinheit realisiert. Normalerweise gibt es keinen Compiler für Modelle ohne Random-Access-Memory.

Um in einer hochleveligen Sprache zu programmieren (Pascal, C, C++), ist es nicht notwendig, sich mit der inneren Arbeitsweise eine Mikrocontrollers zu beschäftigen, da der Compiler in diesem Fall Mehrzweckregister und Befehle selbst

auswählen kann. Es kann jedoch nur von Vorteil sein, zu wissen was in einem Mikrocontroller abläuft. Es ist notwendig die Befehle des Mikrocontrollers genau zu kennen, wenn man zeitkritische Applikationen entwickelt, bei denen die Operationen in einer begrenzten Anzahl an Taktzyklen beendet sein müssen.

3.4 Taktgeber

Wie die meiste digitale Elektronik, arbeitet auch der AVR in einer konstanten Frequenz. Eine konstante Frequenz versichert die Zuverlässigkeit von Datenaustausch im Gerät. Es gibt unterschiedliche Methoden um ein Taktsignal für den AVR zu generieren.

Interner RC Oszillator

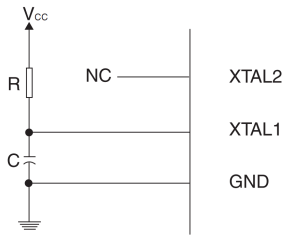


Abbildung 3.17: Using an RC oscillator

Dies ist ein interner Taktgeber, welcher keine externen Komponenten benötigt. Seine größten Nachteile sind niedrige Taktfrequenz und Ungenauigkeit.

Externer RC Oszillator

Funktioniert nach dem gleichen Prinzip wie ein interner Taktgeber und hat keine signifikanten Vorteile gegenüber diesem.

Schwingquarz

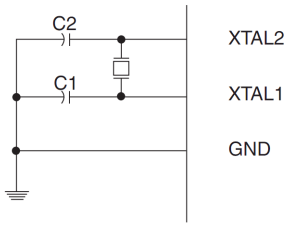


Abbildung 3.18: Using a crystal oscillator

Schwingquarze nutzen einen Kristall (gewöhnlich einen Quarz), welcher in einem elektrischen Feld in seiner Resonanzfrequenz vibriert. Er hat die piezoelektrische Eigenschaft bei mechanischer Deformation (Vibration) ein elektrisches Feld zu produzieren. Schwingquarze haben eine Präzision von ca. 0.001%, welche temperaturunabhängig ist.

Keramische Resonatoren

Keramische Resonatoren ähneln den Schwingquarzen, sind aber aus günstigeren piezoelektrischen Materialien gemacht. Keramische Resonatoren sind in der Regel kleiner als Schwingquarze, aber weniger genau (0.5%) und weniger stark temperaturabhängig.

Externes Taktsignal

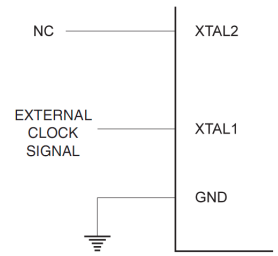


Abbildung 3.19: Using an external clock signal

Ein Externes Taktsignal kann mit jedem Gerät erzeugt werden, solange die Frequenz und die Amplitude (Spannung) im richtigen Bereich liegen. Zum Beispiel kann ein externer Taktgeber benutzt werden um ein Taktsignal gleichzeitig an mehrere Microcontroller weiterzugeben.

3.5 Interrupts

Interrupts können im AVR - je nach Controller - durch Zähler, Kommunikations-Interfaces, Analog-zu-Digital Konverter, Komparator, spezielle Input-Output Pins und verschiedene andere Funktionen hervorgerufen werden. Jeder Interrupt kann von der Einheit die ihn generiert, entweder zugelassen werden oder nicht. Unabhängig davon, ob der Interrupt zugelassen ist oder nicht, gibt es ein 1-Bit Feld (Interrupt Flag) in der dazugehörigen Einheit im Controller, welches mit „true“markiert wird, wenn die Interrupt-Bedingung erfüllt ist. Wurde die Interrupt-Flag auf „true“geändert wird und der Interrupt zugelassen, dann wird der Controller den für den Interrupt gedachten Code ausführen.

Jeder Interrupt des AVR Microcontrollers ist an ein bestimmtes Ereignis gebunden. Jedes Ereignis hat einen Flag-Bit im Status-Register, welcher den Eintritt des Ereignisses markiert. Ereignisse sind ebenfalls an die Register der Interrupt-Masken sowie die dazugehörigen Bits gebunden. Wenn das Interrupt-Bit des Ereignisses unmaskiert ist und ein Ereigniss stattfindet, stoppt der Prozessor die Ausführung des laufenden Programms für einige Arbeitszyklen und führt das Interrupt-Programm aus. Nachdem das Interrupt-Programm ausgeführt wurde, fährt der Prozessor mit seinem pausierten Hauptprogramm fort.

Example

Um Interrupts mit der AVR LibC Bibliothek zu nutzen, ist es notwendig interrupt.h einzubinden. Der Code der ausgeführt wird, wenn der Interrupt stattfindet, wird nach einem „ISR“-Keyword geschrieben. Der Text innerhalb der Klammern nach „ISR“kennzeichnet den Namen des Interrupts. Ein Beispiel in C:

```
#include <avr/interrupt.h>

ISR(XXX_vect)
{
  // Anweisungen
}
```

Die generelle Erlaubnis sämtlicher Interrupts wird vom Control- und Statusregister SREG konfiguriert. Die Option, alle Interrupts sofort zu erlauben oder zu verbieten existiert zum Schutz von Daten. Da Interrupts das Ausführen ei-

nes Programms unterbrechen, könnten einige Daten im Hauptprogramm dabei zerstört oder beschädigt werden. Solche Situationen können einfach vermieden werden, wenn man alle Interrupts verbietet, bevor man mit empfindlichen Daten arbeitet. Das generelle Verbot von Interrupts ist einfach, da hierzu lediglich ein Register (SREG) geändert werden muss. Nachdem der kritische Teil des Programms ausgeführt wurde, können Interrupts einfach wieder zugelassen werden wodurch dann sämtliche Interrupts die zurückgehalten wurden, ausgeführt werden.

Example

Angenommen, in diesem Programm wird eine 16-Bit Variable verwendet, welche sowohl von dem Hauptprogramm, als auch vom Interrupt geändert wird und der Wert dieser Variable wird später einer anderen Variable gegeben.

```
#include <avr/interrupt.h>

// Globale 16-Bit Variable x und y
unsigned short x, y;

// Ein zufälliger Interrupt der den Wert von x ändert.
ISR(XXX_vect)
{
  x = 0x3333;
}

int main()
{
  // x einen Wert geben
  x = 0x1111;

  // Interrupts generell erlauben
  sei();

  // y erhält Wert von x
  y = x;
}
```

Das Programm selbst ist sehr einfach - zuerst bekommt die Variable x den Wert 0x1111 und später wird dieser an y weitergegeben. Wenn ein Interrupt zwischen diesen beiden Operationen auftritt, bekommt x einen Wert von 0x3333. Logisch gesehen kann y somit zwei mögliche Werte am Ende des Programms haben,

allerdings existiert bei einem 8-Bit AVR noch eine dritte Möglichkeit. Hierzu kommt es, da die 8-Bit Architektur zur Verarbeitung von 16-Bit Daten 2 Taktzyklen braucht. Daher kann ein Interrupt durch schlechtes Timing die Integrität dieser Daten beschädigen. Dadurch kann y am Ende des Programms den Wert 0x1111 oder 0x3333, aber auch 0x3311 haben Um den dritten und ungewollten Wert zu verhindern, sollten alle Interrupts temporär verboten werden wenn eine Operation mehr als einen Taktzyklus umfasst.

Im folgenden Beispiel wird eine sichere Methode benutzt um y den Wert von x zu geben:

```
// Interrupts generell verbieten
cli();

// y erhält Wert von x
y = x;

// Sämtliche Interrupts wieder zulassen
sei();
```

3.6 Digitale Inputs/Outputs

Sämtliche Busse des AVR sind sowohl lesbar als auch beschreibbar, wenn sie im normalen logischen Input/Output (I/O) Modus verwendet werden. AVR Busse werden nacheinander mit lateinischen Buchstaben (A, B, C usw.) benannt. Allerdings können einige AVR zwar einen Bus B, jedoch keinen Bus A besitzen. Jeder Bus ist ein 8-Bit Bus und jedes Bit hat normalerweise seinen eigenen Pin am Controller. Pins werden mit Ziffern, beginnend bei Null, versehen. Für beide Richtungen im Bus gibt es ein separates Register. Zusätzlich gibt es noch ein weiteres Register welches die tatsächliche Richtung des Bus festlegt, ein Bitwert von 1 markiert den Bus als Output und ein Wert von 0 als Input. Damit hat der Bus drei Register:

PORT - Zur Festsetzung des Outputwertes

PIN - Zum Auslesen des Inputs am Bus

DDR - Zur Festlegung der Richtung des Busses.

Beispiel

Aufgabe: Die Pins 0-3 von Bus B sollen Input sein, die Pins 4-7 Output, setzen Sie Pin 5 „high“und lesen Sie die Werte der Pins 0-3 in einer Variable aus. Der C-Code sieht wie folgt aus:

```
#include <avr/io.h>

int main()
{
  unsigned char x;

  // Pins 0-3 als Input, 4-7 als Output
  DDRB = 0xF0;

  // Setzt Pin 5 "high"
  PORTB |= (1 << PIN5);

  // Liest den Wert der Pins 0-3
  x = PINB & 0x0F;
}
```

In diesem Beispiel, wird der Input im Hi-Z (hochohmig) Modus verwendet. Im Wesentlichen liegt an diesem Input keine Ladung an. Diese Betriebsart könnte hilfreich sein, wenn der Pin als Datenbus genutzt wird. Es ist sinnvoll, einen Pull-Up-Widerstand für den Input zu nutzen, wenn der Pin als Taste, Schalter oder anderes verwendet wird wobei er mit der Masse verbunden ist. Hierfür muss das korrespondierende Output-Bit im Inputmodus „high“gesetzt werden. Als Folge daraus muss zwischen Stromversorgung und Input ein Widerstand eingefügt werden, welcher die Inputspannung hoch hält, bis irgendetwas sie heruntersetzt. Ein solcher Pull-Up-Widerstand dient dazu, ein Floaten des Inputs aufgrund von Reibungselektrizität oder anderen Einwirkungen zu verhindern. Nachdem der Controller gebootet wurde, befinden sich sämtliche IO Busse standardmäßig im hochohmigen Inputmodus.

Normalerweise werden die Pins am IO Bus, neben den logischen Anschlüssen, auch für andere Peripherieschnittstellen genutzt. Soll eine alternative Funktion

des Pins genutzt werden, kann der zugehörige IO Pin-Modus im AVR Datenblatt abgelesen werden. Wenn beispielsweise ein ADC Kanal als Input dienen soll, sollte der Pin im Input Modus sein. Um jedoch ein ein PWM Signal zu generieren, sollte er im Output Modus betrieben werden. Es gibt jedoch auch Peripheriemodule, die den IO Pin-Modus eigenständig auswählen.

3.7 Externe Interrupts

Externe Interrupts sind eine der einfachsten Funktionen der Peripheriegeräte. Normalerweise haben AVR's 1 bis 8 spezielle Pins, welche benutzt werden um Interrupts im Programm auszulösen, wenn sich entweder ihr logischer Wert ändert oder sie einen bestimmten Status erreichen. Da diese Funktion normalerweise gebraucht wird um externe logische Signale zu überwachen, werden diese Pins auch externe Interrupt-Pins genannt.

Um einen externen Interrupt zu nutzen, muss der Pins als Standard IO Input (er kann auch als Output genutzt werden, jedoch kann der Interrupt dann nur vom Controller selbst ausgelöst werden) konfiguriert werden. Außerdem muss der Empfang von Interrupts zugelassen sein und die Bedingung, welche den Interrupt auslöst, muss im externen Konfigurationsregister des Interrupts spezifiziert werden.

Es existieren vier mögliche Bedingungen:

Logische Null (Spannung von 0 V)

Änderung des logischen Werts

Fallende Flanke - logische Änderung von 1 zu 0

Steigende Flanke - logische Änderung von 0 zu 1

Der Modus „logische Null“bewirkt, dass der Interrupt durchgehend ausgelöst wird, solange der Pin den Wert Null hat. Während dieser Periode wird die Ausführung des Hauptprogramms gestoppt.

Prinzipiell gibt es zwei Typen von Interrupts: Mit dem Taktgeber des Controllers synchrone und asynchrone Interrupts. Synchrone Interrupts funktionieren indem sie den Wert des Inputs behalten. Auf diese Weise können Änderungen der logischen Werte aufgedeckt werden, indem die Werte zwischen zwei Taktzyklen verglichen werden. Erfolgen die Änderungen des Signals schneller als der Tastzyklus des Controllers, können Interrupts falsch oder gar nicht ausgelöst werden. Asynchrone Interrupts sind unabhängig vom Taktgeber des Controllers und können so schnellere Änderungen im externen Signal erfassen - das logische Level muss jedoch immer noch für mind. 50 ns konstant sein. Der ATmega128 hat 4 synchrone und 4 asynchrone externe Interrupts.

Beispiel

Aufgabe: Lassen Sie ATmega128 Pin Nummer 9 (Pin 7 an Bus E) einen Interrupt auslösen, wenn sich sein Wert ändert. Der Pin gehört zum externen, synchronen INT7 Interrupt.

```
#include <avr/interrupt.h>

// Der Code des externen Interrupts
ISR(INT7_vect)
{
  // mache irgendetwas
}

int main()
{
  // Ändere Pin 7 an Bus E zu einem Inputpin, indem Bit 7 auf 0 gesetzt
     wird
  DDRE &= ~(1 << PIN7);

  // Definiere einen Pull-Up Widerstand für Pin 7 an Bus E
        // um Input Floating zu verhindern.
  PORTE |= (1 << PIN7);

  // Setzt den Interruptmodus auf "Logische Änderung" für den Interrupt
     7
        // im externen Interruptkonfigurationsregister.
  EICRB = (1 << ISC70);

  // Erlaube externen Interrupt 7
  EIMSK |= (1 << INT7);
```

```
  // Erlaube globale Interrupts
  sei();

  // Endlosschleife
  while (1) continue;
}
```

Verfügt der AVR über ausreichend Pins, ist es möglich, zusätzlich zu einem Interrupt der von einem einzelnen Pin ausgelöst wird, eine ganze Gruppe an Pins zu nutzen, um Interrupts durch logische Wertänderungen auszulösen. Diese Interrupts werden einfach „Pin change Interrupts“genannt. Sie lösen aus, sobald mindestens der Wert von einem Pin in der Gruppe sich ändert.

3.8 Analog-zu-Digital Konverter

Analog-zu-Digital Konverter (ADC) transformieren einen analogen Spannungswert in einen digitalen Wert. Die erlaubte Spannungsreichweite am ADC-Input eines AVR Microcontrollers liegt bei 0-5.5V. Der digitale Wert ist 10 Bit groß und bis auf $\pm$ 2 Einheiten genau. Der Fehler kann jedoch auch noch größer sein, wenn die Betriebsspannung nicht vor Störungen geschützt ist. Der AVR hat separate Betriebs- und Vergleichsspannungs-Pins für den ADC. Die separate Betriebsspannung trägt dazu bei, Störungen möglichst klein zu halten wobei sie sich nicht mehr als 0.3V von der Hauptbetriebsspannung unterscheiden darf. Die Vergleichsspannung definiert den maximalen digitalen Wert. Beträgt sie zum Beispiel 3V beträgt, dann wird ein Input mit der gleichen Spannung mit 210 - 1 (1023) ausgelesen.

Der AVR ADC arbeitet nach dem Prinzip der schrittweisen Annäherung. Kurz gesagt, die gemessene Spannung wird mit bestimmten Spannungslevels verglichen und die Ergebnisse werden als Bit-Array ausgegeben. Diese Methode ist relativ langsam, da jedes Bit im Endergebnis einzeln berechnet wird. Der AVR benötigt 13 Taktzyklen für jede Messung, für die Erste jedoch (beim Start-up) werden, 25 Taktzyklen benötigt. Diese Taktzyklen entsprechen jedoch nicht den Arbeitstakten des Controllers, sondern sind spezielle Taktzyklen die der ADC-Einheit durch den Frequenzteiler zugewiesen wurden. Um die maximale Präzision zu erreichen, sollte die ADC-Frequenz sollte 50-200 kHz betragen, bei höheren

Frequenzen sinkt die Genauigkeit. Eine höhere Frequenz ist dann sinnvoll, wenn viele Daten ausgelesen werden sollen wobei die Genauigkeit dieser vernachlässigt wird. Gemäß dem AVR-Handbuch dauert eine Messung 13-260 μs.

Der gemessene Wert kann als 8- oder 10-Bit Wert gelesen werden. Da der AVR ein 8-Bit Mikrocontroller ist, verfügt er über zwei 8-Bit Register um ADC-Werte zu speichern. In den Einstellungen kann festgelegt werden, ob die ersten oder letzten zwei Bits in ein seperates Register sollen. Falls die zwei zuletzt ausgelesenen Bits, welche das Ergebnis weniger widergeben, separiert werden, kann das Ergebnis als 8-Bit Wert ausgelesen werden. Eine solche Kombination nennt man „left aligned result“. Darüber hinaus kennzeichnet „right aligned result“jene Kombination, bei der beide Register gelesen werden, und der Wert ein 10-Bit Format hat.

Ein normaler AVR verfügt über 8 analoge Spannungs-Input-Kanäle. Die Mikrocontroller der ATtiny Serie besitzen weniger, einige ATmega haben 16, allerdings existiert immer nur ein ADC. Um verschiedene Inputs nutzen zu können hat das Gerät einen eingebauten Multiplexer. Der Input des Multiplexer kann durch Nutzung eines speziellen Registers definiert werden. Die ADC Einheit weist darüber hinaus noch weitere Eigenschaften auf: Der Sleepmodus des Prozessors wird für die Umwandlung genutzt, um Störungen zu verringern. Außerdem gibt es die Option zur Nutzung einer internen festen Vergleichsspannung (gewöhnlich 2.65 V, bei manchen Modellen aber 1 V)

Beispiel

Aufgabe: Messen Sie die Spannung des ADC Kanals 3 eines ATmega128. Der Spannungsbereich liegt bei 0-5 V und das Ergebnis sollte 8-Bit präzise sein.

```
#include <avr/io.h>

int main()
{
  unsigned char result;

  // Wähle den AREF Pin zum Vergleich der Spannung
  // (es wird angenommen, dass AREF mit der +5V Stromversorgung
    verbunden ist))
  // Wähle Kanal 3 im Multiplexer r
  // ordne das Ergebnis links an
  ADMUX = (1 << REFS0) | (1 << ADLAR) | (3);
```

```
  // Starte den ADC,
  // setze den Umwandlungszyklus 16 mal langsamer als das Tastverhä
    ltnis
  ADCSRA = (1 << ADEN) | (1 << ADPS2) | (1 << ADSC);

  // Warte auf Beendigung des Messvorganges
  while (ADCSRA & (1 << ADSC)) continue;

  // Lies den 8-Bit Wert aus
  result = ADCH;
}
```

3.9 Counters/Timers

Counter, welche man auch in einigen Fällen Taktgeber nennen kann, sind einer der wichtigsten Unterfunktionen eines Microcontrollers. Diese ermöglichen es Prozesse exakt zu timen, Signale zu generieren und Ereignisse zu zählen. Ein Zähler konvertiert die Nummer der Inputzyklen in einen binären Wert mit Hilfe eines Arrays an Triggern. Die maximale Nummer der gezählten Zyklen hängt von der Länge des Arrays ab und diese ist markiert von der Länge des binären Codes. Der AVR hat 8- und 16- Bit Counter. Wenn ein Timer seinen maximalen Wert erreicht hat(255 bei 8-Bit und 65535 bei 16-Bit), generiert der nächste Zyklus ein Overflow und der Zähler macht ein Reset auf 0. Das Taktsignal eines Counters kann vom Taktsignal des Microcontrollers kommen und in diesem Fall ist es möglich den Wert mit einem Prescaler herunter zusetzen. Einige AVRs haben einen internen unabhängigen Taktgeber, welcher mit einem Frequenzmultiplikator modifiziert werden kann, um schneller zu laufen.Counter unterscheiden sich auch in Applikationsfällen und Arbeitsmodi.

Counter's default mode

Im Default-Modus macht ein Counter nichts anderes, als fortlaufend
Nummer zu zählen. Sein wert kann natürlich jederzeit ausgelesen, und von ei-

nem Programm verändert werden. Die einzige zusätzliche Funktion im Default-Modus ist, dass er ein Interrupt beim Counter-Overflow verursacht. Der Default-Modus wird normalerweise benutzt, um eine Sektion eines Programms in bestimmten Intervallen laufen zulassen.

Example

Aufgabe: Ein 8 MHz ATmega128 soll einen Interrupt alle 10ms (100 Hz) auslösen. Dafür ist der 8-Bit Counter 0 brauchbar.

```
#include <avr/interrupt.h>

ISR(TIMER0_OVF_vect)
{
  // Gibt den Counter einen Wert,
  // dass der nächste Overflow in 10Hz passiert.
  // Formel: 256 - 8 MHz / 1024 / 100 Hz = 177,785 = ~178
  TCNT0 = 178;
}

int main()
{
  // Um den ersten Interrupt in 10ms auszulösen,
  // muss der Counter initialisiert werden.
  TCNT0 = 178;

  // Prescaler Wert 1024
  TCCR0 = 0x07;

  // Erlaubt Overflow Interrupts
  TIMSK |= (1 << TOIE0);

  // Erlaubt global Interrupts
  sei();

  // Endlosschleife
  while (1) continue;
}
```

Der Counter in diesem Beispiel wird den Interrupt nicht in exakt 10 ms generieren, dafür müsste man dem Counter einen Dezimalwert geben, das ist aber nicht möglich.Um einen präzisen Intervall zwischen den Interrupts zu verwirklichen, wird sowohl der Prescaler Wert, als auch der Wert des Counter müssen

so ausgesucht werden, dass ihre Division in einer exakten Nummer endet. Das ist nicht immer möglich, vor allem mit 8-Bit Countern, da ihr Wertebereich sehr klein ist. Um einen präziseren oder größeren Intervall zu bekommen, kann ein 16-Bit Counter gewählt werden.

Externer Taktgeber Es ist auch möglich einen externen Taktgeber als das Taktsignals eines Counters zu nutzen.Der AVR hat einen Pin „Tn“
dafür. Das n ist Platzhalter für die Nummer des Counters.Das Externe Taktsignal und die Polarität kann man im Prescaler Register auswählen.

Timing events Da die Counter zeitlich abgestimmte Operationen ermöglichen, haben komplexe AVR Microcontroller eine Option auf
Hardware-Level für zeit spezifische Ereignisse.Der Teil des Counter heißt „Input Capture Unit“. Es gibt eine Auswahl zwischen zwei Ereignissen: Die logische Änderung eines Werts an einem speziellen Input Pins oder im Wert des analogen Komparator Ergebnisses. Wenn das gewählte Ereignis eintritt wird der Wert des Counters in ein spezielles Register geschrieben, welches man jederzeit auslesen kann. Wenn das Event länger ist als die Overflowzeit des Counter, muss das Programm die Overflow mit zählen und beim Endergebnis mitberechnen.

Example

Aufgabe: Miss die Frequenz eines Externen 122Hz - 100kHz logischen Rechtecksignals mit einem 8MHz ATmega128. Die Messung soll mit einer 1 Hz-Präzision erfolgen. Das Programm nutzt einen 16-Bit Counter mit einer Input Capture Unit.

```
#include <avr/interrupt.h>

unsigned long frequency;

// Interrupt für das Ereignis
ISR(TIMER1_CAPT_vect)
{
  // Counter to 0
  TCNT1 = 0;
```

```
  // Das Ergebnis ist nur gültig wenn der Counter
  // noch kein Overflow hatte.
  if (!(TIFR & (1 << TOV1)))
  {
    // Berechnen der Frequenz von der Periode
    frequency = (unsigned long)8000000 /
                (unsigned long)ICR1;
  }
  else
  {
    // Frequenz ist weniger als 122Hz
    frequency = 0;

    // Setzt die Counter Overflow Flagge auf 0
    TIFR &= ~(1 << TOV1);
  }
}

int main()
{
  // Registriert eine "rising front", Prescaler Wert auf 1
  TCCR1B = (1 << ICES1) | (1 << CS10);

  // Erlaubt Ereignis-Interrupts
  TIMSK = (1 << TICIE1);

  // Erlaubt global Interrupts
  sei();

  // Endlosschleife
  while (1) continue;
}
```

Das Programm löst jedes mal bei einer „Rising Front“im externen Signal einen Interrupt aus.Während des Interrupts, überprüft das Programm den Counter auf einen Overflow - dass kann passieren wenn die Frequenz des Signals unter 122Hz (8 MHz / 2^{16}) ist und in so einem Fall spiegelt der Wert des Counters die echte Periode nicht mehr wieder. Die Frequenz wird berechnent indem man mit 32-Bit Nummern die Inverse der Periode bekommt. Als erstes setzt man den Counter auf 0, weil der Timer mit dem gleichen Taktsignal wie der Prozessors arbeitet und jede Befehlsausführung nach dem externen Event die zu messende Periode verkürzt und damit das Ergebnis verfälscht. Die maixmale gemessene

Frequenz wird durch die Zeit die das Interrupt benötigt limitiert.

Events erfassen und die Zeit registrieren die es brauchte damit das Event auftrat kann auf Software Level abgehandelt werden. Es ist möglich einen externen oder andere Interrupts zu nutzen und den Wert des Counters während des Events auszulesen. Das Erfassen auf Hardwarelevel wird genutzt um unabhängig vom Hauptprogramm zu arbeiten und sehr kurze (oder häufige) Events zu timen.

Taktsignal generieren

Komplexere Counter können ein Taktsignal generieren, zusätzlich zum Timen eines Signals. Dafür hat der Counter eine „Output Compare Unit“
und eine „Compare Match Output Unit“. Die Output-Compare-Unit hat Register mit der gleichen Bit-Weite wie der Counter und die Werte dieser Register werden mit den Werten des Counters verglichen während dieser läuft.Ein Interrupt kann generiert werden und spezielle Pin Werte können geändert werden wenn der Counter Wert gleich der des Compare Unit Registers ist. In diesem Moment kann ein Pin entweder High oder low gesetzt oder invertiert werden. Das Signal wird durch Änderungen im Wert des Output-Pins generiert.

In einigen Signal-generierenden Modi, kann der maximale Wert des Counters verändert werden.Die physikalische Größe bleibt die gleiche, aber ein Vergleichsregister wird benutzt um den Counter bei einem bestimmten Wert zu resetten.Das vorherige Beispiel kann auch mit dieser Methode gelöst werden, aber die Funktion ist eher für das Ändern der Periode des Signals gedacht. Zusätzlich kann ein Counter so eingestellt werden, dass er mit auf- oder absteigenden Werten arbeitet.

Der Counter und die Signal-generierenden Modi die diese nutzten sind einer der komplexesten Peripheriemodule in einem AVR. Über jeden hier zu schreiben würde eine große Zeitverschwendung sein, normalerweise gibt es keinen Grund alles zu wissen, um sie zu nutzen. Der folgende Absatz beschreibt einer der üblichen PWM Signale in den Robotics. Der Rest kann den AVR Dokumentationen entnommen werden.

Pulse Width Modulation Pulsweitenmodulation (PWM) ist ein Typ eines Signals, wo die Frequenz und die Periode(normalerweise) konstant sind, aber die Länge der Halb-Periode ändert sich. PWM Signale werden benutzt um elektromechanische, optische und andere Geräte zu kontrollieren.Zum Beispiel, ein Servomotor nutzt ein ein 50Hz PWM Signal und haben einen hohe Halbperiode von 1 - 2 ms.

Example

Aufgabe: Generiere mit einem 8Mhz ATmega128, zwei Geschwindigkeits-regulierende Servomotor Signale. Nutze Pin PB5 (OC1A) um eine Pulsweite von 1ms zu generieren und Pin PB6 (OC1B) um eine Pulsweite von 2ms zu generieren.

```
#include <avr/io.h>

int main()
{
  // Set pins as outputs
  DDRB |= (1 << PIN5) | (1 << PIN6);

  // Set outputs A and B low for comparison,
  // "Fast PWM" mode, prescaler value 8
  TCCR1A = (1 << COM1A1) | (1 << COM1B1) | (1 << WGM11);
  TCCR1B = (1 << WGM13) | (1 << WGM12) | (1 << CS11);

  // Maximum value of the counter. Formula:
  //   TOP = 8 MHz / 8 / 50 Hz
  ICR1 = 20000;

  // Half-period of the first motor is 1 ms, and second 2 ms
  OCR1A = 1000;
  OCR1B = 2000;

  // Endless loop
  while (1) continue;
}
```

3.10 USART

USART ist ein Universal Synchrones Serielles Interface, UART ist die vereinfachte Version - Universelles Asynchrones Interface. Der Unterschied zwischen beiden liegt darin, dass das USART auch eine Taktsignalleitung nutzt um Daten zu synchronisieren, während das UART nur Datenleitungen benutzt. Das UART des AVR Mikrocontrollers Duplexverkehr, 5- bis 9-Bit Datenworte (8-Bit Wort = 1 Byte), 1 oder 2 Stop Bits, 3 Paritätsmodi und eine Vielzahl an Baud-Raten zulässt. AVR Mikrocontroller verfügen normalerweise über bis zu 2 USART Interfaces, jedoch gibt es auch einige ohne USART. Datenübertragungen werden mit einen Wort für Wort ausgeführt - der AVR konvertiert das Wort, welches er vom Benutzer bekommt in Bits auf Hardwareeinheiten und übermittelt es unabhängig und andersherum. Der Benutzer kann das USART mit Schreib- und Lesekonfigurationen sowie Status- und Datenregistern kontrollieren.

Jede Einstellungsoption hat ein eigenes Register, welches recht einfach anhand des Datenblatts konfiguriert werden kann. Die Baud-Rate ist allerdings etwas schwieriger einzustellen. Das Taktsignal für die Datenübertragung wird vom Taktgeber des Controllers generiert und der Nutzer kann eine Zahl von 1 bis 4096 eingeben, durch welche die Taktrate dividiert wird. Das Ergebnis wird darüber hinaus, je nach Modus, durch 2, 8 oder 16 dividiert. Das Problem ist, dass nicht alle Taktfrequenzen so dividiert werden können, dass das Ergebnis eine Standard-Baud-Rate ergibt. Bei einigen Frequenzen kann die Baud-Rate bis zu 10% vom Standardwert abweichen. Die Datenblätter des AVR beinhalten Tabellen mit Angaben zu den typischen Taktfrequenzen, Baud-Raten, den nötigen Multiplikatoren um die Baud-Raten zu erreichen sowie den möglichen Rechenfehlern.

Da die Datenübertragung unabhängig vom Prozessor und viel langsamer geschieht, ist es nötig zu bestätigen, dass das Interface für das nächste Wort bereit ist, bevor eine weitere Übertragung stattfindet. Das kann gewährleistet werden, indem das Ready Bit des Sendepuffers beachtet wird, welches anzeigt ob der Puffer bereit ist, ein neues Wort zu empfangen oder nicht. Der Controller startet mit einen aktivierten Ready Bit. Sobald ein Wort übertragen wird und der Puffer leer ist, wird das Ready Bit „high“gesetzt.

Der Empfang eines Wortes ist wird durch ein spezielles Status-Bit gekennzeichnet. Zusätzlich gibt es Status-Bits, welche Rahmenfehler, Paritätsfehler und Puf-

ferüberläufe anzeigen. Ein Pufferüberlauf tritt auf, wenn das letzte Wort noch aus dem Puffer gelesen werden muss, während ein neues ankommt. Aus diesem Grund ist es wichtig, die empfangenen Datenwörter stets so schnell wie möglich ins Programm zu lesen, z.B. mit Hilfe eines Interrupts. Es gibt drei mögliche Gründe für Interrupts: Sendepuffer bereit, Übertragung erfolgreich, Empfang erfolgreich.

Der Sende- und Empfangspuffer sind physikalisch getrennte Register, aber sie nutzen die gleiche Speicheradresse und den gleichen Namen. Während des Schreibens in ein gemeinsam benutztes Register, werden die Daten im Sendepuffer gespeichert, beim Auslesen werden sie aus dem Empfangspuffer gelesen. Werden 9-Bit Worte benutzt, wird das 9. Bit mit Hilfe eines Einstellungsregister übertragen und ausgelesen.

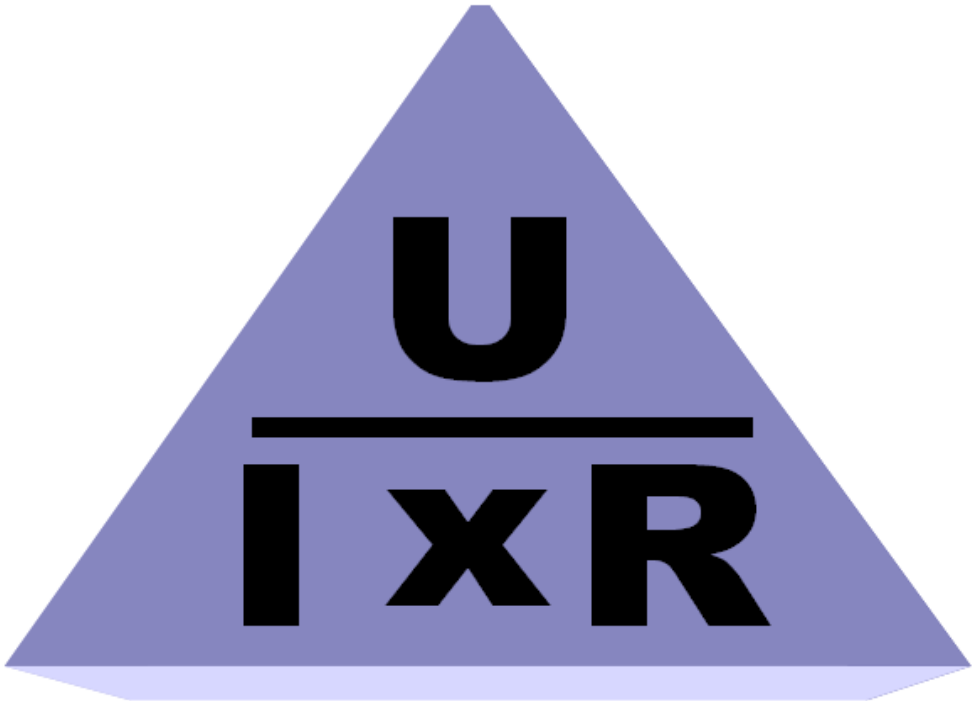
Beispiel

Aufgabe: Konfigurieren Sie die USART0 Schnittstelle eines 8 MHz ATmega128 so, dass sie ein 8-Bit-Wort asynchron überträgt, wobei die Baudrate 9600 bps beträgt, 1 Stoppbit und kein Paritätsbit genutzt werden soll. Senden Sie das Symbol “X“.

```
#include <avr/io.h>

int main()
{
  // Setzt die Baud-Rate auf 9600 bps. Formel:
  //   Faktor = Frequenz / 16 / Baudrate -1
  //   UBRR = 8000000 / 16 / 9600 - 1 = ~51
  UBRR0H = 0;
  UBRR0L = 51;

  // Erlaube Senden
  UCSR0B = (1 << TXEN0);

  // Konfiguriere den asynchronen Modus, setze die Wortgröße auf 8 Bit
    ,
  // 1 Stoppbit, kein Paritätsbit.
  UCSR0C = (1 << UCSZ01) | (1 << UCSZ00);

  // Warte solange, bis der Datenpuffer leer ist (das vorherige Wort
    wurde gesendet)
  // In diesem Beispiel ist es jedoch nicht nötig zu warten, da nur das
```

```
    erste Symbol
        // gesendet wird. Es sollte jedoch beachtet werden, wenn mehr
    als ein Symbol gesendet wird.
  while (!(UCSR0A & (1 << UDRE))) continue;

  // Schreibe das Symbol in den Sendepuffer.
  UDR0 = 'X';

  // Endlosschleife
  while (1) continue;
}
```

4 Elektronik



Verschiedene elektronische Schaltkreise werden so häufig in praktischen Beispielen verwendet, dass sie in den folgenden Kapiteln separat beschrieben werden. Da sie das Verständnis der Beispiele sowie den Bau eigener Schaltkreise erleichtern, ist es wichtig sie zu kennen.

4.1 Ohm'sches Gesetz

Das Ohm'sche Gesetz besagt, dass die Stromstärke eines elektrischen Leiters direkt proportional zu der potenziellen Differenz oder Spannung zwischen zwei Punkten sowie umgekehrt proportional zu dem Widerstand zwischen diesen ist, unter der Voraussetzung, dass die Temperatur konstant bleibt.

Die Gleichung lautet wie folgt:

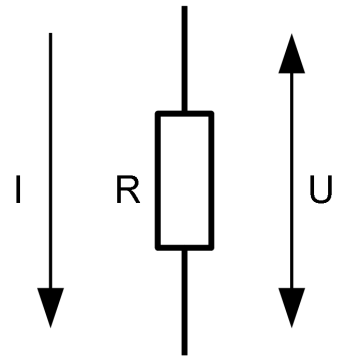


Abbildung 4.1: Resistance of the conductor, applied voltage and current through the resistance

I = U / R

mit:

I für die Stromstärke [in A]

U für die Spannung [in V]

R für den elektrische Widerstand [in Ω]

Viele weitere, in der Elektronik häufig verwendete Gleichungen sind vom Ohm'schen Gesetz abgeleitet.

4.2 Berechnung des LED-Widerstands

Für den Fall, dass die Betriebsspannung höher ist als die Spannung der LED, muss der Widerstand in Reihe in den Stromkreislauf geschaltet werden. Er begrenzt die Spannung und erzeugt dadurch den notwendigen Spannungsabfall. Um den korrekten Widerstand auszuwählen, wird folgende, vom Ohm'schen Gesetz abgeleitete, Gleichung angewandt:

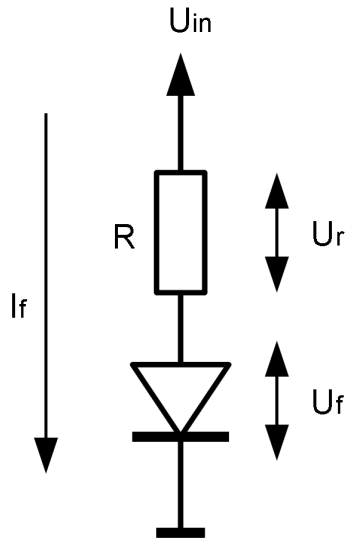


Abbildung 4.2: LED mit in Serie geschaltetem Widerstand

$R = (U_{in} - U_f) / I_f$
$U_r = U_{in} - U_f$
$P_r = U_r \cdot I_f$

mit:

R für den Widerstand.

U_{in} für die Betriebsspannung.

U_f für die LED Spannung.

I_f für die LED Stromstärke.

U_r für den Spannungsabfall am Widerstand.

P_r für die Leistung des Widerstands, welche in Wärme umgewandelt wird.

Der spannungsbegrenzende Widerstand einer LED darf nicht geringer sein als R und muss mindestens die Leistung von P_r haben.

4.3 Spannungsteiler

Ein Spannungsteiler ist ein elektrischer Schaltkreis, der die Eingangsspannung teilt. Die Ausgangsspannung ist ein Teil der Eingangsspannung und der Wert ist abhängig vom Verhältnis der Widerstände im Schaltkreis. Die Abbildung rechts zeigt ein Beispiel mit zwei Widerständen.

Die Ausgangsspannung des Schaltkreises wird mit folgender Gleichung berechnet:

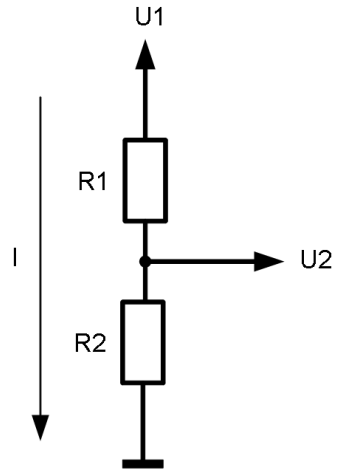


Abbildung 4.3: Schaltplan eines Spannungsteilers

$U_2 = U_1 \cdot (R_2 \ / \ (R_1 + R_2))$

mit:

U_1 für die Eingangsspannung

U_2 für die Ausgangsspannung

R_1 und R_2 für die Widerstände

Die Gleichung ist vom Ohm'schen Gesetz abgeleitet.

mit:

$I = U_1 \ / \ (R_1 + R_2)$

und:

$U_2 = I \cdot R_2$

Spannungsteiler werden häufig im Zusammenhang mit Widerstandssensoren (Fotowiderstand, temperaturgesteuerter Widerstand etc.) verwendet, wobei einer der Widerstände durch einen Widerstandssensor ersetzt wird. Dadurch wird die Widerstandsänderung des Sensors einfach in eine Spannungsänderung transformiert, welche kompatibel zu dem analogen Input des Mikrocontrollers ist.

5 Robotic HomeLab Kit

Das auf dem ATmega128 Mikrocontroller basierende Mechatronik und Robotik HomeLab ist eine Zusammenstellung von Modulen in einem tragbaren Koffer. Es ermöglicht die Durchführung diverser Mechatronik- und Robotikexperimente und -übungen, angefangen bei einem Blinklicht bis hin zur Erstellung komplexer Geräte. In erster Linie richtet sich das HomeLab an Schulen, da es zusätzlich zur Hardware methodologisches Material sowie Übungen mit Musterlösungen enthält. Das HomeLab wurde in eine Web-Umgebung integriert, welche Schülern und Lehrern die gegenseitige Interaktion ermöglichen soll. Kurz gefasst ist das HomeLab kit eine Sammlung mobiler Werkzeuge mit dem Zweck zu Hause, in der Schule oder am Arbeitsplatz lernen und üben zu können.

Das HomeLab kit wurde von der Tallinn University of Technology sowie estnischen Unternehmen in Kooperation mit europäischen Partneruniversitäten und der Unterstützung des Leonardo da Vinci Programms entwickelt. Die HomeLab Module sind zu verschiedenen Bausätzen zusammengestellt. Die Grundausstattung, welche zum Erlernen einfacher digitaler In- und Output-Operationen dient, nennt sich HomeLab Basic kit. Der Sensoren und Bedienteil Ergänzungsbausatz besteht aus einer Zusammenstellung von Sensoren, Motoren sowie deren Treibern. Das Advanced kit enthält sowohl das HomeLab Basic kit sowie das Sensors and Actuators Add-on kit und darüber hinaus zusätzliche Module.

Die HomeLab Basic und Advanced kits eignen sich zur Durchführung von Experimenten, als Basisplattform für Roboter sowie andere mechatronische Regeltechniken. Das HomeLab wird ständig weiterentwickelt, daher lohnt es sich, regelmäßig nachzuschauen, ob neue Übungsaufgaben oder Modul-Updates veröffentlicht wurden. Darüber hinaus ist es lohnenswert, die Versionsnummern der Module zu überprüfen, da einige Übungsaufgaben und Beispiele in neueren Versionen verfügbar sind. Die Nutzung von DistanceLab erhöht den Nuten von HomeLab maßgeblich. DistanceLab ermöglicht es, die HomeLab Hardware über das Internet zu betreiben.

HomeLab Webseite
http://home.roboticlab.eu

DistanceLab Webseite
http://distance.roboticlab.eu

5.1 Ganzheitliches Konzept der Robotik

Die Ingenieurausbildung kann sich heutzutage nicht mehr auf einer bestimmten Lehrmethode aufbauen, wie zum Beispiel klassische universitäre Vorlesungen und Übungen. Lernende sind heute nicht mehr mit zeitlich festgelegten Vorlesungen und Übungen zufrieden. Viele junge Leute, aber auch Erwachsene, leben ein Leben in einer modernen Informationsgesellschaft, in welchem Sie viel Zeit im Internet verbringen. Daher bestätigen die klassischen Lernmethoden die Lernenden und ihre Art zu Lernen heute nicht mehr. Aufgrund dessen müssen die Ingenieursausbildungen in eine Internetumgebung gebracht und dadurch attraktiver werden, ohne jedoch die Qualität und die praktische Erfahrung zu verlieren.The integrated concept of robotic and mechatronic study incorporates the standard teaching and studying aids and novel approaches, which are integrated into one methodical unit.

Das ganzheitliche Konzept der Robotik und Mechatronik beinhaltet sowohl standardisierte Lehrmethoden und Lernhilfen, als auch neue Ansätze, welche in einer systematischen Einheit untergebracht sind.

Das folgende Beispiel kann Schulen helfen Kurse zur Robotik zu starten oder einen Denkanstoß zu geben, wie man verschiedene Lernkonzepte in einer praktische Lerneinheit in diesem technologischen Feld anwendet.

Das Konzept beinhaltet die folgenden Lehr- und Lernhilfsmittel:

Koventionelles Lehrbuch, z.B. „Integrated Systems & Design“ISBN: 978-9985-59-850-4.

Praktisches Übungsbuch

Robotic HomeLab

DistanceLab

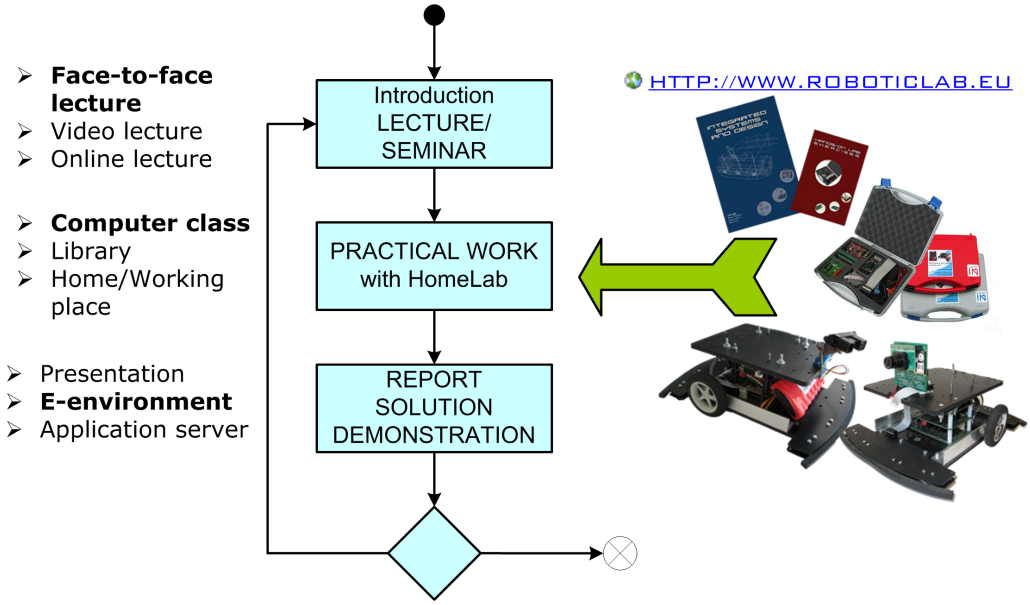
Das Kompetenznetzwerk der Robotik- und Mechatronikgemeinschaft http://www.roboticlab.eu.

Theoretische Lernhilfsmittel sind das konventionelle Lehrbuch und das Übungs-

buch, welches zusätzlich zur Papierversion auf elektronisch erhältlich ist.

Der praktische Teil besteht aus dem Robotic HomeLab und dem DistanceLab, einer internetbasierten Programmierumgebung für Roboter. Das HomeLab bildet zusammen mit dem DistanceLab eine kompakte Lösung, was bedeutet, dass der Lernende auch einzelne Funtionen der Robotik (z.B. Sensormesswerte, Motorsteuerung, etc.) von zuhause aus ausprobieren und gleichzeitig sein erlerntes Wissen im Robotersystem anwenden kann. Das Robotersystem kann über das DistanceLab erreicht werden, welches aus realen Geräten besteht. Das System selbst ist aus denselben Komponenten zusammengestellt, wie sie auch im mobilen
HomeLab-Koffer enthalten sind.

Der Ablauf von einem Lernprozess bzw. der Arbeit mit dem Lab ist auf dem nächsten Bild beschrieben.



Das Thema beginnt mit einer Einführung, welche eine klassische Vorlesung, eine Onlinevorlesung oder eine Videovorlesung sein kann. Die Videovorlesungen sind auch für die klassischen Vorlesungen wichtig, da sie die Möglichkeit bieten eine Vorlesung erneut zu sehen, sollte etwas unklar geblieben oder der Studierende nicht in der Lage gewesen sein, an der Vorlesung teil zu nehmen. Die

Einführungsvorlesung gibt eine Übersicht über das jeweilige Problem und mit Hilfe eines Lehrers kann dann ein praktisches Beispiel durchgegangen werden. Der theoretische Teil wird durch das Lehrbuch und das praktische Übungsbuch unterstützt.

Nach der Einführungsvorlesung wird der praktische Arbeitsteil realisiert. Die Praktische Arbeit besteht anfangs darin, die Codebeispiele auszuführen und später darin, die Beispiele entsprechend der Übung zu verändern. Daraufhin folgen komplexere Übungen, welche durch einen Lehrer jeder Person oder jedem Team individuell aus dem Übungsbuch zugeteilt werden.

Die Ergebnisse der praktischen Arbeit werden in einem Bericht zusammengefasst und dem Betreuer zugeschickt. Falls die Entwicklungsumgebung eine automatische Berichterstattung unterstützt, so kann diese anstatt dem Senden einer Email genutzt werden. Zusätzlich zu den Textberichten wird außerdem die HEX-Datei für die Arbeitsmaschine eingebunden. Der Bericht umfasst typischerweise die Beschreibung der Arbeit und einen vollständig kommentierten Quellcode der Lösung.

5.2 Robotik unterrichten

Es ist allseits bekannt, dass die Informations- und Telekommunikationstechnologie (ICT) in das tägliche Leben von vielen Menschen eingetreten ist und die Art und Weise des Lernens und der Kommunikation verändert hat. Viele junge Leute sind es gewohnt Informationen über das Kursmaterial über verschiedene Kommunikationswege aus dem Internet zu bekommen. Bevor versucht wird ein Problem zu lösen wird erst eine schnelle Suche nach möglichen Lösungen, Beispielen oder Hintergrundinformationen durchgeführt. Da die tägliche Kommunikation teilweise in soziale Netzwerke verlagert wurde, ist es nur natürlich, dass die Methodik, welche von den Schulen angewandt wird auch diesem Trend folgen muss, um die Studierenden in ihrem täglichen Umfeld zu erreichen. Natürlich wird die traditionelle Lehrmethodik nicht komplett ersetzt, aber sich nur auf diese zu verlassen ist heutzutage nicht mehr genug.

Robotik und Mechatronik sind sehr zukunftsreiche Gebiete, in denen es sehr wichtig ist den neuesten Trends und Technologien zu folgen. Gleichzeitig sind

diese Gebiete sehr praktisch veranlagt und es verlangt viel manuelle Erfahrung Fähigkeiten und Wissen zu erlangen. Selbst verschiedene virtuelle Simulationen können durchgeführt werden um das Systemverhalten zu verstehen. Trotzdem ersetzten die virtuellen Simulationen nicht die praktische Erfahrung mit echter Hardware.Das Folgende ist eine praktische Anleitung wie ein Robotikkurs eingeführt werden kann, in dem verschiedene Lehransätze genutzt werden. Die Methodik erwartet, dass der Lehrer das HomeLab kit benutzen kann.

Aspekte die bei der Erstellung eines neuen Kursen berücksichtigt werden müssen:

1. **Gruppengröße**
   Die optimale Anzahl der Teammitglieder ist weitgehend durch die Anzahl der Computer Arbeitsplätze begrenzt. Das bedeutet, dass normalerweise nicht mehr als 3 Personen zusammen an einem Arbeitsplatz arbeiten können. Wenn mehr Personen an einem Arbeitsplatz arbeiten müssen könnte es sein, dass sie nicht in der Lage sind aktiv an der Arbeit teilzunehmen und somit die Konzentration auf die Arbeit verloren geht.

2. **Praktische Arbeit und Berichterstattung**
   Die Praktische Arbeit ist unterteilt in Übungseinheiten, wobei jede Übung eine ähnliche Struktur besitzt und mit einem Bericht endet, jedoch unterschiedliche Fähigkeiten trainiert. Es ist außerdem möglich die Arbeit innerhalb der Gruppe aufzuteilen, so dass zum Beispiel ein Mitglied für die Erstellung des Quellcodes, ein anderes Mitglied für die Erstellung des Berichts und ein Mitglied für das Anschließen der Hardware verantwortlich ist. Zusätzlich zu dem kommentierten Quellcode sollte der Bericht auch Algorithmen, Beschreibungen der Arbeit, Antworten auf die gestellten Fragen, ein Schema der Hardwareverbindungen und Vor- und Nachteilbeurteilungen der Übung enthalten. Der letzte Punkt ist ein gutes Feedback für den Lehrenden um den Schwierigkeitsgrad der Übung einzuschätzen und wie interessant die Übung für die Lernenden war.
   Es ist wichtig, dass man die Beispiele zusammen mit einem Betreuer implementiert, da dieser Hintergrundinformationen zu den Funktionen bieten und die Beispiele lösen kann. Die ganze Vorgehensweise sollte schriftlich Schritt für Schritt festgehalten werden, damit jeder Lernende folgen kann und so sicher gegangen werden kann, dass alle Teilnehmer den Ab-

lauf verstanden haben, auch wenn sie in unterschiedlichen Geschwindigkeiten voran kommen. Eine mögliche Übungsablaufstruktur ist im Folgenden beschrieben, kann jedoch je nach Situation und Bedürfnissen angepasst werden.

Einführung in die Übung

- Der Lehrer erklärt die neue Übung, geht alle nötigen theoretischen Inhalte durch und führt zusammen mit dem Teilnehmern ein Beispielprogramm aus.

- Die Übungseinheit ist in kleinere Teile unterteilt, wobei jeder Teil aus einem theoretischen und einem praktischen Teil besteht, in welchem dann Quellcode entwickelt werden soll. Das bedeutet, dass der Quellcode anhand von theoretischen Erklärungen entwickelt wird.

Individuelle Arbeit. Die individuelle Arbeit erfolgt gemäß den Arbeitsanweisungen der Übungseinheit. Hier sind Ziele, Arbeitsprozesse, Anforderungen und die Berichterstattung definiert. Die Schüler lernen selbstständig (entweder alleine oder in einer kleinen Gruppe) und erhalten, wenn nötig, Unterstützung von ihren Lehrern. Ziel des eigenständigen Arbeitns ist jedoch, die Lösung selbstständig zu finden. Wenn möglich, sollten die Schüler das HomeLab kit mit nach Hause nehmen können, um so auch dort arbeiten zu können. So sind sie nicht an den Zeitrahmen und Ort der Übung gebunden.

Berichte. Lernende müssen einen Bericht verfassen, in welchem sie ihre Arbeit gemäd́en Anforderungen an einen Bericht beschreiben und welcher Antworten auf Fragen des Lehrers an die Gruppe gibt. Der Bericht sowie die Lösung der Aufgabe (HEX-Datei) sollen dem Lehrer per E-Mail zugeschickt oder in den E-Learning Bereich hochgeladen werden.

Überprüfung

- Der Lehrer überprüft und bewertet den Bericht sowie die erarbeitete Lösung jeder Gruppe qualitativ.

– Er kann überprüfen, ob die Lösung entsprechend der Übungsvoraussetzungen funktioniert und darüber hinaus von den Schülern verlangen, das Programm mündlich zu erläutern. In diesem Fall sollte jedes Gruppenmitglied einen Teil des Berichtes erklären.

3. **Gruppenmanagement**
   Bei der Durchführung von Übungen in einem Klassenraum kommt es häufig vor, dass eine GRuppe langsamer arbeitet als die anderen und somit stärker vom Lehrer unterstützt werden muss, um nicht hinterherzuhängen. Ähnlich ist es, wenn eine Gruppe schneller arbeitet als alle anderen und somit auf die nächste Aufgabe warten muss. Beide Probleme können minimiert werden, wenn der Arbeitsprozess für alle offensichtlich dargestellt ist und jeder einzelne die vollständigen Übungsanweisungen zu Beginn der Übung erhält. So kann die Gruppe arbeiten, ohne auf die nächstesn Anweisungen des Lehrers warten zu müssen, indem sie einfach den Anweisungen folgt. Darüber hinaus enthält jeder Übungsabschnitt am Ende Aufgaben in verschiedenen Schwierigkeitsgraden, die der Lehrer je nach Fortschritt und Fähigkeiten der Gruppe verteilen kann.

4. **Bewertung**
   Soll letztendlich sicher gestellt werden, dass alle Teilnehmer die vorgestellten Programmierfähigkeiten erlernt haben, so kann der Leherer die Gruppenarbeit derart gestalten, dass jedes Gruppenmitglied selbst programmieren und dieses auch präsentieren muss. Eine andere Möglichkeit wäre, dass der Lehrer zufällig ein Mitglied der Gruppe auswählt, welches dann das Programm vorstellen muss. Auch wenn diese Person den Code vielleicht nicht persönlich geschrieben hat. Das Endergebnis wird dann für die gesamte Gruppe gezählt, ausgehend vom Schwächsten Mitglied. Sollte diese Art der Bewertung ungerecht wirken, kann auch innerhalb der Gruppe eine individuelle Berichterstattung durchgeführt werden.

5. **Fehlerbehebung**
   Die Schüler machen im Umgang mit der Hardware oder beim Programmieren häufig ähnliche Fehler. Daher ist es sinnvoll, auf der Webseite einen Bereich für häufig gestellte Fragen (FAQ - Frequently Asked Questions) einzurichten. So können die Lernenden Zeit sparen, da sie bei häufig auf-

tretenden Fragen oder Problemen schnell Hilfe finden. Der FAQ-Bereich sollte zusammen mit den Lösungen im Support-Bereich der HomeLab Webseite verfügbar sein.

6. **Wettbewerb**
   Zur Steigerung der Begeisterung für Robotik-Studien kann die praktische Arbeit mit einer Art Wettbewerb verbunden werden. Die Gruppen müssen zu einem gegebenen Problem eine Lösung erarbeiten, welche anschließend gemeinsam bewertet wird. Die Beurteilungskriterien (Zeit, Geschwindigkeit, Entfernung, etc.) müssen klar definiert und bekannt gegeben werden. Darüber hinaus können weitere Kriterien herangezogen werden um die Ansicht, den Erfindungsreichtung, die Kosteneffizenz und weitere Aspekte der Lösung zu bewerten. Die Aufgabe kann darüber hinaus auch mit Problemstellungen des täglichen Lebens verbunden werden wie zum Beispiel ein Projekt zu Batteriesortierern oder Sicherheitssystemen. Meistens werden Aufgaben aus der Hobbyrobotk oder der mobilen Robotik gestellt, wie beispielsweise ein Roboter, der einer Linie folgen soll, Sumoroboter, Ballspiele etc.

Das folgende Beispiel veranschaulicht eine typische Übungsanweisung, die für praktische Übungen verwendet werden kann.

Lab Nr 3. Sensoren und Analog-Digital Konverter

Leiter: Sven Seiler

Das Ziel

1. Grundlagen über Analog-Digital-Konverter und das Konvertieren analoger Signale mit dem AVR ATmega128 8-Bit Mikrocontroller.

2. Verschiedenen Sensoren mit analogem Output kennenlernen

3. Eine einfache Übung mit einem analogen Sensor durchführen.

Benötigte Ausrüstung

HomeLab Basic kit, Sensor and Motor Add-On kit, AVR Programmiersoftware.

Arbeitsanweisungem

1. Führen Sie die Übung mit dem Potentiometer durch.
   http://home.roboticlab.eu/en/examples/sensor/potentiometer

2. Führen Sie die allgemeine Aufwärmübung durch.
   http://home.roboticlab.eu/en/exercises/sensor

3. Absolvieren Sie die gruppenspezifische Aufgabe. (diese wird rechtzeitig vom Übungsleiter bekannt gegeben)

4. Beantworten Sie die Fragen (gruppenspezifische Fragen werden rechtzeitig vom Übungsleiter bekannt gegeben)

Bericht

Der elektronische Bericht muss nach der eigenständigen Arbeit eingereicht werden und folgende Struktur aufweisen:

Arbeitsbericht

- Das Ziel
- Kurze Beschreibung der durchgeführten Arbeit
- Ausdruck des Algorithmus und Quellcodes für die Schritte 2 und 3. Achtung! Der Quellcode muss kommentiert und farbig gekennzeichnet sein(um das Farbsystem korrekt in MS Word oder einem OpenOffice Programm darzustellen könnte das „Programmer's Notepad"genutzt werden)
- Anworten auf die Fragen (Schritt4)
- Ergebnisse und Kommentare

Lösungen (HEX-Dateien) von Schritt 2 und 3

Der Bericht muss den eigenen Namen, die Nummer des Übungsabschnitts, das Datum sowie die Namen der Gruppenmitglieder (bei einer Gruppenarbeit) ent-

halten. Er sollte kurz und präzise verfasst sein wobei die Qualität wichtig ist, nicht die Quantität. Sie sollten darauf vorbereitet sein, die Lösung zu präsentieren oder den Quellcode zu kommentieren. Der Bereicht muss in den Bereich für Hausarbeiten des E-Learning Systems hochgeladen werden. Der letzte Abgabetermin für den Bericht ist eine Woche nach Beginn der Übung.

Lektüren

1. HomeLab Support-Center: http://home.roboticlab.eu
2. ATmega128 Datenblatt
3. Sharp Infrarotsensor Datenblatt
4. http://www.avrfreaks.net

5.3 HomeLab Hardware

Die Module des Robotik HomeLab sind in unterschiedliche Bausätze („kits“) eingeteilt, um so die am besten geeignete Ausstattung für die Bedürfnisse eines Studenten oder einer Schule auswählen zu können. Das einfachste Set ist das HomeLab Basic kit, welches die für grundlegende Mikrocontrollerexperimente notwendigen Komponenten enthält. Darüber hinaus verfügt es über eine Anwendungsoberfläche mit LCD Display. Das Basic kit kann zusammen mit dem Motor and Sensor Add-On kit verwendet werden. Letzteres beinhaltet verschiedene Arten von Motoren und Sensoren sowie zusätzliches Zubehör und Kommunikationsmodule. Das Sensor and Motor Add-On kit kann nicht ohne das Basic kit verwendet werden, da der Mikrocontroller nicht in dem Add-On enthalten ist. Außerdem enthält das Add-On kit RFID sowie Machine Vision Module. Basic und Add-On kit sind komfortabel im Gebrauch und können bequem transportiert werden, um die Robotikexperimente zu Hause oder am Arbeitsplatz durchzuführen.

HomeLab Basic kit

Abbildung 5.1: HomeLab Basic kit

AVR ATmega2561 Entwicklungsplatine, inklusive Ethernet, SD Kartenlesen und integriertem JTAG Programmiergerät

Benutzerschnittstellenplatine (Schalter, LEDs, graphisches LCD, 7-Segment Anzeige)

USB Kabel

Beispielaufgaben mit C-Quellcode

Softwarebibliothek

Stromversorgung

Multimeter

Software zur Programmierung in Assembler und C

Tragbarer Koffer

Sensor and Motor Add-On kit

Sensormodul

Analoge Sensoren- und Tiefpassplatine mit on-board Sensoren (Tempera-

tursensor, Lichtsensor, Potentiometer und Mikrofon)

Ultraschallentfernungssensor SRF05 mit Kabel

Infrarotentfernungssensor mit Kabel

Abbildung 5.2: Sensor and Motor Add-On kit

Motormodul

DC Motor (mit Getriebe und Kodierer)

RC Servomotor

Schrittmotor (uni- oder bipolarer Schrittbetrieb)

Motorentreiberplatine

Versorgungsteiler

Kommunikationsmodul

Kommunikationsplatine, 2xRS232

ZigBee oder Bluetooth Funkeinheit

Zusätzliche HomeLab Module

Zusätzliche Module sind nicht in den Koffern enthalten, können jedoch direkt mit dem HomeLab Kommunikationsmodul verbunden werden. Es gibt für diese Module ebenfalls praktische Beispiele und Übungsaufgaben.

RFID Modul

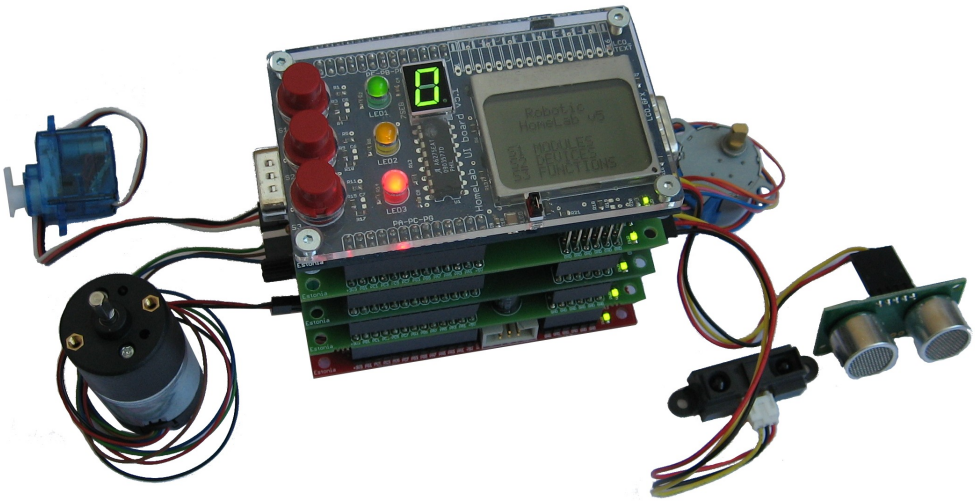
Hochfreqzenz RFID Lesegerät

Tag (2)

Kabel

Machine Vision Module

Intelligentes Kamera Module (CMUcam3)



— MISSING PAGE —— MISSING PAGE —— MISSING PAGE —— MISSING PAGE —

5.4 HomeLab Bibliothek

Die HomeLab Bibliothek besteht aus einigen C Header-Dateien (mit der Dateiendung „.h“) und einer statischen Bibliotheksdatei (mit der Dateiendung „.a“). Während der Installation der Bibliothek werden diese Dateien vollständig in den AVR-GCC Unterordner kopiert. Hier findet der Compiler sie automatisch. Der Benutzer muss diese Dateien somit nicht mehr in den Programmordner kopieren.

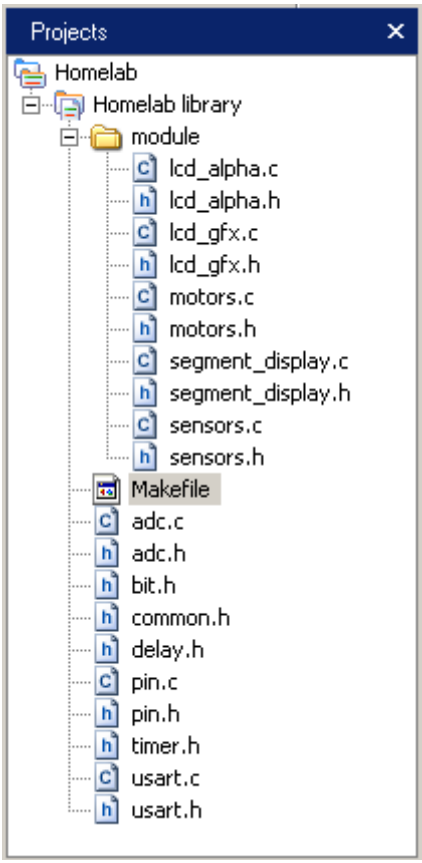


Abbildung 5.3: HomeLab Bibliothek Quellcode Dateien

Die schrittweisen Installationsanweisungen der AVR Entwicklungssoftware für Windows und Linux sind im ersten Kapitel zu den praktischen Beispielen dargestellt. Auch wenn unterschiedliche Beispiele auf verschiedene Teile der Bibliothek zurückgreifen, muss in jedem Projekt die statische Bibliotheksdatei („.a“Datei) vorhanden sein. Nur die Header-Dateien können wahlweise eingeladen werden. Header-Dateien, die in direktem Bezug zum AVR Mikrocontroller stehen, sind im „homelab“-Ordner abgelegt. HomeLab modulspezifische Dateien sind im Ordner „homelab/module“zu finden. Beide Ordner befinden sich im Root-Ordner des Compilers. Um die Dateien dieser Ordner zu verwenden, müssen größer- / kleiner-Zeichen genutzt werden um den Pfad anzugeben. Nachfolgend finden Sie ein Beispiel, wie Header-Dateien für AVR Pins und HomeLab Motoren integriert werden können:

```
#include <homelab/pin.h>
#include <homelab/module/motors.h>
```

Falls die HomeLab Bibliothek nicht genutzt wird, muss die folgende avrlibc Header-Datei in das Projekt eingefügt werden:

```
#include <avr/io.h>
```

Die HomeLab Bibliothek enthält diese Datei bereits in der *pin.h* Datei.

Das Installationsprogramm für die HomeLab Bibliothek ist auf der HomeLab Webseite verfügbar. Nutzer, die die Bibliothek an Ihre Bedürfnisse anpassen möchten, können hier auch den Quellcode dafür herunterladen. Das folgende Kapitel beschreibt die Funktionen der Bibliothek.

5.4.1 Bitweise Operationen

Die Bibliothek für bitweise Operationen beinhaltet eine Zusammenstellung von Makrofunktionen zur Durchführung von Bitmanipulationsoperationen. Sie werden von den anderen Komponenten der Bibliotheken genutzt und können überall angewandt werden. Da die Makrofunktionen keinen speziellen Typ haben, sind sie mit jedem Datentyp kompatibel.

Der Bit-Index wird genutzt um das Bit in einer Binärzahl zu spezifizieren. Indizes werden beginnend bei Null gezählt, wobei Null das niedrigstwertige Bit (LSB) kennzeichnet. Eine 8-Bit-Zahl hat zum Beispiel Indizes von 0 bis 7, eine 16-Bit Zahl Indizes von 0 bis 15.

Funktionen

bit_mask(bit)
Bit-Index zu Bitmaske konvertieren. Parameter:

- *bit* - Bit-Index.
- Gibt Bitmaske aus.

bit_set(value, bit)
Setzt ein bestimmtes Bit in einer Variablen. Parameter:

- *value* - Variable.
- *bit* - Bit-Index.

bit_clear(value, bit)
Löscht ein bestimmtes Bit in einer Variablen. Parameter:

- *value* - Variable.
- *bit* - Bit-Index.

bit_set_to(value, bit, state)
Setzt ein bestimmtes Bit einer Variablen in einen gewünschten Zustand. Parameter:

- *value* - Variable.
- *bit* - Bit-Index.
- *state* - Status (*true* or *false*).

bit_invert(value, bit)
Invertiert ein bestimmtes Bit der Variable. Parameter:

- *value* - Variable.
- *bit* - Bit-Index.

bit_is_set(value, bit)
Überprüft, ob ein bestimmtes Bit in einer Variablen gesetzt ist oder nicht. Parameter:

- *value* - Variable.
- *bit* - Bit-Index.

– Gibt den Bool'schen Wert *true*, wenn das Bit gesetzt und *false* wenn das Bit gelöscht wurde zurück.

bit_is_clear(value, bit)
Überprüft, ob ein bestimmtes Bit in einer Variablen gelöscht wurde oder nicht. Parameter:

– *value* - Variable.

– *bit* - Bit-Index.

– Gibt den Bool'schen Wert *true*, wenn das Bit gelöscht wurde und*false* wenn das Bit gesetzt wurde zurück.

Beispiel

Das dritte Bit einer 8-Bit Variablen *b* setzen und Invertieren des letzten Bits.

```
#include <homelab/bit.h>

int main(void)
{
  unsigned char b = 0x00;

  bit_set(b, 2);
  bit_invert(b, 7);
}
```

Quelle

Nachfolgend eine gekürzte Version des Quellcodes der Bibliothek für bitweise Operationen.

```
//
// Funktionen zur Bearbeitung von Bits.
//
#define bit_mask(bit)           (1 << (bit))
#define bit_set(value, bit)     value |= bit_mask(bit)
#define bit_clear(value, bit)   value &= ~bit_mask(bit)
```

```
#define bit_invert(value, bit)    value ^= bit_mask(bit)
#define bit_is_set(value, bit)    ((value) & (bit_mask(bit)))
#define bit_is_clear(value, bit) (!((value) & (bit_mask(bit))))
#define bit_set_to(v, b, x) v = ((x) ? (v | bit_mask(b)) : (v & ~
    bit_mask(b)))

//
// Funktionen zur Bearbeitung von Bitmasken.
//
#define bitmask_set(value, bitMask)      value |= (bitMask)
#define bitmask_clear(value, bitMask)    value &= ~(bitMask)
#define bitmask_invert(value, bitMask)   value ^= (bitMask)
#define bitmask_set_to(v, m, x)          v = ((x) ? (v | (m)) : (v & ~(m
    )))
#define bitmask_is_set(value, bitMask)   ((value) & (bitMask))
```

5.4.2 Pins

Die Pins-Bibliothek erleichtert die Arbeit mit digitalen AVR In- und Output-Pins. Der Nutzer kann eine zu einem Pin gehörige Variable erstellen und mit dieser sämtliche Pin-Operationen vornehmen. Auf diese Weise entfällt die Verwendung von Registernamen oder Bit-Indizes, wie es beim Programmieren mit direktem Zugriff auf das Register nötig ist. Port und Index des Pins müssen nur einmalig festgelegt werden, sodass Änderungen leicht implementiert werden können.

Datentypen

pin Datentyp für Adressen von Pin-Registern und Bitmasken. Um das effizienteste Programm zu entwickeln, sollten *pin* Variablen konstant sein und am Anfang des Programmcodes initialisiert werden. Letzteres kann mit der Makrofunktion *PIN* durchgeführt werden, deren erster Parameter der Buchstabe des Ports ist (A, B, C, etc.) und zweiter Parameter der Pin-Index (0 - 7). Es können nur bestehende Ports und Pins verwendet werden.

Funktionen

void pin_setup_output(pin pin)
Konfiguriert einen Pin als Output. Parameter:

- *pin* - Pin Variable.

void pin_setup_input(pin pin)
Konfiguriert einen Pin als Input ohne Pull-up Widerstand. Parameter:

- *pin* - Pin Variable.

void pin_setup_input_with_pullup(pin pin)
Konfiguriert einen Pin als Input mit Pull-up Widerstand. Parameter:

- *pin* - Pin Variable.

void pin_set(pin pin)
Setzt einen Output-Pin high. Parameter:

- *pin* - Pin Variable.

void pin_clear(pin pin)
Setzt einen Output-Pin low. Parameter:

- *pin* - Pin Variable.

void pin_toggle(pin pin)
Invertiert den Status eines Output-Pins. Parameter:

- *pin* - Pin Variable.

void pin_set_to(pin pin, bool value)
Setzt einen Output-Pin in den gewünschten Zustand. Parameter:

- *pin* - Pin Variable.

- *value* - Gewünschter Status als Bool'scher Wert.

bool pin_get_value(pin pin)

Holt Pin-Wert. Parameter:

- *pin* - Pin Variable.
- Gibt Bool'schen Wert *true* wenn Pin high und *false* wenn Pin low ist aus.

bool pin_get_debounced_value(pin pin)
Liest den Pinwert durch den Switch-Debounce-Filter. Das Filtern benötigt mindestens 8 ms und kann bis zu 100 ms benötigen, je nachdem wann das Bouncing endet. Falls es nicht endet wird *false* ausgegeben. Funktionen nutzen Softwareverzögerung. Parameter:

- *pin* - Pin Variable.
- Gibt dem Pin den Bool'schen Wert *true* wenn der Pin high ist und *false* wenn der Pin low oder unbestimmt ist.

Beispiel

Beispiel um einen Pin-Wert zu erhalten und zu setzen. Pin PC0 Wert wird invertiert und an Pin PC3 angeschlossen.

```
#include <homelab/pin.h>

pin output_pin = PIN(C, 3);
pin input_pin = PIN(C, 0);

int main(void)
{
  bool value;

  // Konfiguriert Pin als Output-Pin
  pin_setup_output(output_pin);

  // Konfiguriert Pin als Input-Pin mit Pull-up Widerstand
  pin_setup_input_with_pullup(input_pin);

  // Endlosschleife
  while (true)
  {
```

```
    // Wert eines Input-Pins erhalten
    value = pin_get_value(input_pin);

    // Wert für den Output-Pin setzen
    pin_set_to(output_pin, !value);
  }
}
```

5.4.3 Analog-Digital-Wandler

Diese Bibliothek enthält Funktionen um den AVR Analog-Digital-Wandler zu nutzen. Sämtliche Funktionen dieser Bibliothek sind blockierend, d.h. der Prozessor wartet so lange, bis die Ergebnisse da sind. Die Zeit für die Konvertierung hängt von der ADC-Taktung ab.

Datentypen

adc_reference
ADC Vergleichsspannung, Aufzählungsdatentyp. Möglichkeiten:

- *ADC_REF_AREF* - Vergleichsspannung vom AREF Pin.
- *ADC_REF_AVCC* - Vergleichsspannung vom AVCC Pin.
- *ADC_REF_2V56* - Interne 2,56 V Vergleichsspannung.

adc_prescale
ADC Takt-Vorteiler, Aufzählungsdatentyp. Der Vorteiler gibt den Divisionsfaktor für den Systemtakt an. Möglichkeiten:

- *ADC_PRESCALE_2* - Divisionsfaktor 2.
- *ADC_PRESCALE_4* - Divisionsfaktor 4.
- *ADC_PRESCALE_8* - Divisionsfaktor 8.
- *ADC_PRESCALE_16* - Divisionsfaktor 16.

- *ADC_PRESCALE_32* - Divisionsfaktor 32.
- *ADC_PRESCALE_64* - Divisionsfaktor 64.
- *ADC_PRESCALE_128* - Divisionsfaktor 128.

Funktionen

void adc_init(adc_reference reference, adc_prescale prescale)
Initialisiert ADC. Parameter:

- *reference* - Auswahl der Vergleichsspannung.
- *prescale* - Auswahl des Takt-Vorteilers.

unsigned short adc_get_value(unsigned char channel)
Konvertiert bestimmte analoge ADC Kanalwerte in digitale. Die Funktion blockiert. Parameter:

- *channel* - ADC Kanalnummer (0 bis 7).
- Liefert einen digitalen 10-bit Wert.

unsigned short adc_get_average_value(unsigned char channel, unsigned char num_samples)
Konvertiert eine gewünschte Anzahl analoger Werte eines bestimmten ADC Kanals zur digitalen Werten und berechnet den Mittelwert. Die Funktion blockiert.

- *channel* - ADC Kanalnummer (0 bis 7).
- *num_samples* - Anzahl der Werte für die Berechnung (1 bis 64).
- Liefert einen digitalen 10-bit Wert.

Beispiel

Der ADC sei initialisiert und zwei analoge Kanalwerte zu digitalen konvertiert. Der Wert von Kanal 0 wird der Variable x zugewiesen und der durchschnittliche Wert von Kanal 1 der Variable y.

```
#include <homelab/adc.h>

int main(void)
{
  unsigned short x, y;

  // Initialisierung ADC. Vergleichsspannung von AVCC.
  // Takt ist 8 mal langsamer als der Systemtakt.
  adc_init(ADC_REF_AVCC, ADC_PRESCALE_8);

  // Konvertierung des Wertes von Kanal 0.
  x = adc_get_value(0);

  // Konvertieren des Wertes von Kanal 1 und Errechnung des
    Durchschnitts.
  y = adc_get_average_value(1, 10);
}
```

5.4.4 Serielles Interface

Diese Bibliothek bietet die Funktionen für das AVR asynchrone serielle Interface

Datentypen

usart
Datentyp für USART Interfacekontroll-, Status- und Datenregisteradressen. *usart* Variablen sollten zu Beginn des Programms initialisiert werden. Für die Initialisierung gibt es eine Macrofunktion *USART* dessen einziger Parameter für den Index des Interface ist. (0 oder 1).

usart_databits
Zählen der Datenbits, Aufzählungsdatentyp. Möglichkeiten:

- *USART_DATABITS_5* - 5 Datenbits.
- *USART_DATABITS_6* - 6 Datenbits.
- *USART_DATABITS_7* - 7 Datenbits.
- *USART_DATABITS_8* - 8 Datenbits.
- *USART_DATABITS_9* - 9 Datenbits.

usart_stopbits
Zählen der Stop-Bits, Aufzählungsdatentyp. Möglichkeiten:

- *USART_STOPBITS_ONE* - Ein Stop-Bit.
- *USART_STOPBITS_TWO* - Zwei Stop-Bit.

usart_parity
Paritätsmodus, Aufzählungsdatentyp. Möglichkeiten:

- *USART_PARITY_NONE* - Ausgeschaltet.
- *USART_PARITY_EVEN* - Gerade Parität.
- *USART_PARITY_ODD* - Ungerade Parität.

Funktionen

USART_BAUDRATE_ASYNC(baud)
Makrofunktion um den USART Baudratenregisterwert im asynchronen Modus zu berechnen. Parameter:

- *baud* - Gewünschte Baudrate
- Gibt Baudratenregisterwert wieder.

void usart_init_async(usart port, usart_databits data_bits, usart_stopbits stop_bits, usart_parity parity, usart_baudrate baudrate)

Initialisiert asynchrones USART. Parameter:

- *port* - USART Schnittstellenvariable.
- *data_bits* - Datenbits.
- *stop_bits* - Stop-Bits.
- *parity* - Paritätsmodus.
- *baudrate* - Baudratenregisterwert (kann mit der Makrofunktion *USART_BAUDRATE_ASYNC* errechnet werden).

void usart_send_char(usart port, char symbol)
Blockiert die Übertragung von Zeichen. Die Funktionen warten solange bis der Ausgabespeicher leer ist, bevor ein Zeichen in den Speicher geschrieben wird. Parameter:

- *port* - USART Schnittstellenvariable.
- *symbol* - zu sendendes Zeichen.

void usart_send_string(usart port, char *text)
Blockierende String-Übertragung. Parameter:

- *port* - USART Schnittstellenvariable.
- *text* - Zeiger auf einen String(char array). Der String muss mit einer Null enden.

bool usart_has_data(usart port)
Überprüft Empfangsspeicher auf Daten. Parameter:

- *port* - USART Schnittstellenvariable.
- Gibt *true* zurück wenn Zeichen im Empfangsspeicher und *false* wenn nicht.

char usart_read_char(usart port)

Liest ein Zeichen aus dem Empfangsspeicher. Zuvor muss der Nuter überprüfen ob tatsächlich ein Zeichen empfangen wurde. Parameter:

- *port* - USART Schnittstellenvariable.
- Gibt Zeichen aus.

bool usart_try_read_char(usart port, char *symbol)
Liest ein Zeichen aus dem Empfangsspeicher, falls eins vorhanden ist. Parameter:

- *port* - USART Interfacevariable.
- *symbol* - Verweis auf die Variable des Zeichens. Wenn ein Zeichen im Empfangsspeicher vorhanden ist, wird es der betreffenden Variable zugeschrieben.
- Gibt *true* aus, wenn ein Zeichen im Empfangsspeicher vorhanden ist und *false* wenn nicht.

Beispiel

Die USART Schnittstelle ist konfiguriert um 8 Datenbits, ein Stop-Bit, 9600 bps Baudrate und keinen Paritätsmodus zu nutzen. Das Programm sendet einen String, wartet bis die Zeichen empfangen wurden und ließt sie dann aus.

```
#include <homelab/usart.h>

// Nutze USART Schnittstelle 0.
usart port = USART(0);

int main(void)
{
  char c;

  // Initialisierung der Schnittstelle.
  usart_init_async(port,
    USART_DATABITS_8,
    USART_STOPBITS_ONE,
    USART_PARITY_NONE,
```

```
    USART_BAUDRATE_ASYNC(9600));

  // Senden des String.
  usart_send_string(port, "Hello\n");

  // Auf ankommende Daten warten.
  while (!usart_has_data(port)) {}

  // Empfangenes Zeichen auslesen.
  c = usart_read_char(port);
}
```

5.4.5 Timer

Diese Bibliothek deckt einen großen Teil der Funktionen des Timers des ATmega128 ab. Es gibt Datentypen und Funktionen welche das Benutzen von Timern vereinfachen. Aufgrund der Komplexität der AVR Timer gibt es leider keine allgemeingültige Funktion, die für verschiedene Timer genutzt werden kann. Die Funktionen jedes Timers beginnen mit dem Präfix „timer“und seinem Index.

Datentypen

timer0_prescale
Timer 0 Vorteiler, Aufzählungsdatentyp. Möglichkeiten:

- *TIMER0_NO_PRESCALE* - Kein Vorteiler (keine Division).
- *TIMER0_PRESCALE_8* - Clk / 8.
- *TIMER0_PRESCALE_32* - Clk / 32.
- *TIMER0_PRESCALE_64* - Clk / 64.
- *TIMER0_PRESCALE_128* - Clk / 128.
- *TIMER0_PRESCALE_256* - Clk / 256.

– *TIMER0_PRESCALE_1024* - Clk / 1024.

timer2_prescale
Timer 2 Vorteiler, Aufzählungsdatentyp. Möglichkeiten:

– *TIMER2_NO_PRESCALE* - kein Vorteiler (keine Division).

– *TIMER2_PRESCALE_8* - Clk / 8.

– *TIMER2_PRESCALE_64* - Clk / 64.

– *TIMER2_PRESCALE_256* - Clk / 256.

– *TIMER2_PRESCALE_1024* - Clk / 1024.

– *TIMER2_PRESCALE_T2_FALLING* - Taktgeber an Pin T2 fallende Flanke.

– *TIMER2_PRESCALE_T2_RISING* - Taktgeber an Pin T2 steigende Flanke.

timer1_prescale

timer3_prescale
Timer 1/3 Vorteiler, Aufzählungsdatentyp. Möglicheiten („n“bedeutet 1 oder 3):

– *TIMERn_NO_PRESCALE* - Kein Vorteiler (keine Division).

– *TIMERn_PRESCALE_8* - Clk / 8.

– *TIMERn_PRESCALE_64* - Clk / 64.

– *TIMERn_PRESCALE_256* - Clk / 256.

– *TIMERn_PRESCALE_1024* - Clk / 1024.

– *TIMERn_PRESCALE_Tn_FALLING* - Taktgeber an Pin Tn fallende Flanke.

– *TIMERn_PRESCALE_Tn_RISING* - Taktgeber an Pin Tn steigende Flanke.

timer1_ctc_top

timer3_ctc_top
Timer 1/3 CTC Modus, Höchstwert, Aufzählungsdatentyp. Möglichkeiten („n“bedeutet 1 oder 3):

– *TIMERn_CTC_TOP_OCRA* - Höchstwert des Timers 1/3 output compare register A.

– *TIMERn_CTC_TOP_ICR* - Höchstwert des Timers 1/3 input capture register.

timer1_fast_pwm_top

timer3_fast_pwm_top
Timer 1/3 fast PWM mode top value enumeration data type. Options („n“means 1 or 3):

– *TIMERn_FAST_PWM_TOP_256* - Höchstwert 255.

– *TIMERn_FAST_PWM_TOP_512* - Höchstwert 511.

– *TIMERn_FAST_PWM_TOP_1024* - Höchstwert 1023.

– *TIMERn_FAST_PWM_TOP_ICR* - Höchstwert des Timers 1/3 input capture register.

– *TIMERn_PAST_PWM_TOP_OCRA* - Höchstwert des Timers 1/3 output compare register A.

timer1_fast_pwm_output_mode

timer3_fast_pwm_output_mode
Timer 1/3 fast PWM mode outputs configuration enumeration data type. Options („n“means 1 or 3):

- *TIMERn_FAST_PWM_OUTPUT_DISABLE* - No output.
- *TIMERn_FAST_PWM_OUTPUT_TOGGLE_ON_MATCH* - Output toggles on compare match.
- *TIMERn_FAST_PWM_OUTPUT_CLEAR_ON_MATCH* - Output clears on compare match.
- *TIMERn_FAST_PWM_OUTPUT_SET_ON_MATCH* - Output sets on compare match.

Funktionen

void timer0_init_normal(timer0_prescale prescale)
Initialisiert Timer 0 im normalen Modus. In diesem Modus zählt der Timer von 0 bis 255 (inkl.). Overflowinterrupts können genutzt werden. Parameter.

- *prescale* - Prescaler.

void timer2_init_normal(timer2_prescale prescale)
Initialisiert Timer 2 im normalen Modus. In diesem Modus zählt der Timer von 0 bis 255 (inkl.). Overflowinterrupts können genutzt werden. Parameter.

- *prescale* - Prescaler.

void timer0_stop()

void timer2_stop()
Stoppt Timer 0/2.

unsigned char timer0_get_value(void)

unsigned char timer2_get_value(void)
Returns timer 0/2 current value. Parameters:

– Return 8-bit timer value.

void timer0_set_value(unsigned char value)

void timer2_set_value(unsigned char value)
Sets timer 0/2 value. Parameters:

– *value* - New 8-bit timer value.

void timer0_overflow_interrupt_enable(bool enable)

void timer2_overflow_interrupt_enable(bool enable)
Enables or disables timer 0/2 overflow interrupt. The name of the interrupt vector is „TIMERn_OVF_vect“where „n“represents 0 or 2. Parameters:

– *enable* - *true* to enable interrupt, *false* to disable.

bool timer0_overflow_flag_is_set(void)

bool timer2_overflow_flag_is_set(void)
Checks timer 0/2 overflow flag. Parameters:

– Returns *true* when overflow has happened, *false* when not.

void timer0_overflow_flag_clear(void)

void timer2_overflow_flag_clear(void)
Resets timer 0/2 overflow flag.

void timer1_init_normal(timer1_prescale prescale)

void timer3_init_normal(timer3_prescale prescale)
Initializes timer 1/3 in normal mode. In this mode timer counts from 0 to 65535 (including). Overflow interrupt can be used. Parameters:

– *prescale* - Prescaler.

void timer1_init_ctc(timer1_prescale prescale, timer1_ctc_top top)

void timer3_init_ctc(timer3_prescale prescale, timer3_ctc_top top)
Initializes timer 1/3 in CTC (*Clear Timer on Compare Match*) mode. In this mode timer counts to specified top value. Overflow interrupt can be used. Parameters:

- *prescale* - Prescaler.
- *top* - Timer top value selection. Actual value must be specified with register selected as a top value holder.

void timer1_init_fast_pwm(timer1_prescale prescale,
timer1_fast_pwm_top top,
timer1_fast_pwm_output_mode output_a,
timer1_fast_pwm_output_mode output_b, timer1_fast_pwm_output_mode output_c)

void timer3_init_fast_pwm(timer3_prescale prescale,
timer3_fast_pwm_top top,
timer3_fast_pwm_output_mode output_a,
timer3_fast_pwm_output_mode output_b,
timer3_fast_pwm_output_mode output_c)
Initializises timer 1/3 in fast PWM mode. In this mode timer counts to a specified value, which also determines the period of the PWM signal. Timer 1/3 has three 3 output compare units (A, B and C) to generate PWM signals. Overflow and compare match interrupts can be used. Parameters:

- *prescale* - Prescaler.
- *top* - Timer top value selection. Actual value must be specified with register selected as a top value holder.
- *output_a* - Output compare unit A pin configuration.
- *output_b* - Output compare unit B pin configuration.
- *output_c* - Output compare unit C pin configuration.

void timer1_stop()

void timer3_stop()
Stops timer 1/3.

unsigned char timer1_get_value(void)

unsigned char timer3_get_value(void)
Returns timer 1/3 current value. Parameters:

- Returns 16-bit timer value.

void timer1_set_value(unsigned char value)

void timer3_set_value(unsigned char value)
Sets timer 0/2 value. Parameters:

- *value* - New 16-bit timer value.

unsigned short timer1_get_compare_match_unitA_value(void)

unsigned short timer1_get_compare_match_unitB_value(void)

unsigned short timer1_get_compare_match_unitC_value(void)

unsigned short timer3_get_compare_match_unitA_value(void)

unsigned short timer3_get_compare_match_unitB_value(void)

unsigned short timer3_get_compare_match_unitC_value(void)
Returns timer 1/3 output compare unit A/B/C compare match register value. Parameters:

- Returns 16-bit compare match register value.

void timer1_set_compare_match_unitA_value(unsigned short value)

void timer1_set_compare_match_unitB_value(unsigned short value)

void timer1_set_compare_match_unitC_value(unsigned short value)

void timer3_set_compare_match_unitA_value(unsigned short value)

void timer3_set_compare_match_unitB_value(unsigned short value)

void timer3_set_compare_match_unitC_value(unsigned short value)
Sets timer 1/3 output compare unit A/B/C compare match register value. Parameters:

- *value* - New 16-bit compare match register value.

unsigned short timer1_get_input_capture_value(void)

unsigned short timer3_get_input_capture_value(void)
Returns timer 1/3 input capture register value. Parameters:

- Returns 16-bit input capture register value.

void timer1_set_input_capture_value(unsigned short value)

void timer3_set_input_capture_value(unsigned short value)
Sets timer 1/3 input capture register value. Parameters:

- *value* - New 16-bit input capture register value.

void timer1_overflow_interrupt_enable(bool enable)

void timer3_overflow_interrupt_enable(bool enable)
Enables or disables timer 1/3 overflow interrupt. The name of the interrupt vector is „TIMERn_OVF_vect“where „n“represents 1 vÕi 3. Parameters:

- *enable* - *true* to enable interrupt, *false* to disable.

void timer1_compare_match_unitA_interrupt_enable(bool enable)

void timer1_compare_match_unitB_interrupt_enable(bool enable)

void timer1_compare_match_unitC_interrupt_enable(bool enable)

void timer3_compare_match_unitA_interrupt_enable(bool enable)

void timer3_compare_match_unitB_interrupt_enable(bool enable)

void timer3_compare_match_unitC_interrupt_enable(bool enable)
Enables or disables timer 1/3 output compare unit A/B/C compare match interrupt. The name of the interrupt vector is „TIMERn_COMPx_vect“where „n“represents 1 or 3 and „x“represents A, B or C. Parameters:

- *enable* - *true* to enable interrupt, *false* to disable.

void timer1_input_capture_interrupt_enable(bool enable)

void timer3_input_capture_interrupt_enable(bool enable)
Enables or disables timer 1/3 input capture interrupt. The name of the interrupt vector is „TIMERn_CAPT_vect“, where „n“represents 1 or 3. Parameters:

- *enable* - *true* to enable interrupt, *false* to disable.

bool timer1_overflow_flag_is_set(void)

bool timer3_overflow_flag_is_set(void)
Checks timer 1/3 overflow flag. Parameters:

- Returns *true* when overflow has happened, *false* when not.

bool timer1_input_capture_flag_is_set(void)

bool timer3_input_capture_flag_is_set(void)
Checks timer 1/3 input capture flag. Parameters:

- Returns *true* when input capture has done, *false* when not.

void timer1_overflow_flag_clear(void)

void timer3_overflow_flag_clear(void)
Resets timer 1/3 overflow flag.

void timer1_input_capture_flag_clear(void)

void timer3_input_capture_flag_clear(void)

Resets timer 1/3 input capture flag.

Beispiel

Im folgenden Programm wird Timer 0 im normalen Modus mit Overflowinterrupt gestartet.

```
#include <homelab/timer.h>
#include <avr/interrupt.h>

// Overflow interrupt program.
ISR(TIMER0_OVF_vect)
{
}

int main(void)
{
  // Initializing of timer 0 in normal mode.
  timer0_init_normal(TIMER0_PRESCALE_32);

  // Enabling timer 0 overflow interrupt.
  timer0_overflow_interrupt_enable(true);

  // Global interrupts enabling.
  sei();
}
```

5.4.6 Verzögerung

Dieser Teil der Bibliothek enthält die benötigten Funktionen, um in einem Programm mit Hilfe von Software Algorithmen oder Hardwaretimern Verzögerungen zu generieren. Verzögerungen blocken keine Interrupts, daher werden Softwareverzögerungen durch Interrupts unterbrochen. Verzögerungsfunktionen sind nicht vor-kompiliert, sodass sie mit verschiedenen Taktfrequenzen genutzt werden können.

Funktionen

void sw_delay_ms(unsigned short count)
Softwareverzögerung in Millisekunden (ms). Für diese Funktion wird ein Compiler Optimierungsmodus benötigt. Parameter:

- *count* - Verzögerungszeit in ms. 0 bis 65535 ms.

void sw_delay_us(unsigned short count)
Softwareverzögerung in Mikrosekunden (μs). Für diese Funktion wird ein Compiler Optimierungsmodus benötigt. Parameter:

- *count* - Verzögerungszeit in μs. 0 bis 65535 μs.

void hw_delay_ms(unsigned short count)
' Zeitgeber basierte Verzögerung in Millisekunden. Diese Funktionen verwenden den ATmega128 8-bit Timer 0. In Abhängigkeit von der Taktfrequenz kann eine Verzögerung von bis zu einigen Millisekunden auftreten. Parameter:

- *count* - Verzögerungszeit in ms. 0 bis 65535 ms.

Beispiele

Demonstration beider Typen einer Verzögerung:

```
#include <homelab/delay.h>

int main(void)
{
  // Software-basierte Verzögerung von 100 ms.
  sw_delay_ms(100);

  // Hardware- (Timer-) basierte Verzögerung von 100 ms.
  hw_delay_ms(100);
}
```

5.4.7 7-Segment LED Display

In Bezug auf: *User Interface Module*

Diese Bibliothek dient zur Nutzung des 7-Segment LED Displays der digitalen HomeLab digitalen I/O Modulplatine gedacht. Mit dieser Bibliothek können Zahlen von 0 bis 9 dargestellt werden.

Funktionen

void segment_display_init(void)
Konfiguriert Kontrollpins für den Displaytreiber.

void segment_display_write(unsigned char digit)
Zeigt eine Ziffer auf dem Display an. Parameter:

- *digit* - Zahlenwert von 0 bis 9. Ansonsten wird ein E für „Error“angezeigt.

Beispiel

Die Zahl 5 wird auf dem LED-Display angezeigt.

```
#include <homelab/module/segment_display.h>

int main(void)
{
  // Initialisierung des Displays
  segment_display_init();

  // Anzeige der Zahl.
  segment_display_write(5);
}
```

5.4.8 Alphanumerisches LCD

Bezug nehmend auf: *lcd*

Diese Bibliothek enthält die Funktionen, um das alphanumerische LCD des HomeLab zu nutzen.

Datentypen

lcd_alpha_mode
LCD Konfiguration, Aufzählungsdatentyp. Möglichkeiten:

- *LCD_ALPHA_DISP_OFF* - Display aus.
- *LCD_ALPHA_DISP_ON* - Display an mit unsichbarem Cursor.
- *LCD_ALPHA_DISP_ON_CURSOR* - Display an mit Cursor.
- *LCD_ALPHA_DISP_ON_CURSOR_BLINK* - Display an mit blinkendem Cursor.

Funktionen

void lcd_alpha_init(lcd_alpha_mode disp_attr)
Initialisiert LCD. Parameter:

- *disp_attr* - Display Konfiguration.

void lcd_alpha_clear(void)
Löscht die Anzeige. Cursor wird zum Anfang der ersten Zeile bewegt.

void lcd_alpha_clear_line(unsigned char line)
Löscht eine Zeile am Display. Cursor wird zum Anfang der ersten Zeile bewegt. Parameter:

– *line* - Zeilennummer: 0 oder 1.

void lcd_alpha_home(void)
Cursor wird zum Anfang der ersten Zeile bewegt.

void lcd_alpha_goto_xy(unsigned char x, unsigned char y)
Bewegt den Cursor zur gewünschten Position. Parameter:

– *x* - X Koordinate (Spaltennumner). 0 bis 15.

– *y* - Y Koordinate (Zeilennumner). 0 bis 1.

void lcd_alpha_write_char(char c)
Schreibt ein Zeichen auf die Position des Cursors. Parameter:

– *c* - ASCII Zeichen.

void lcd_alpha_write_string(const char *s)
Schreibt einen String aufs Display, beginnend bei der Cursorposition. Parameter:

– *s* - Zeiger auf einen String(char array).

void lcd_alpha_write_string_p(const char *progmem_s)
Schreibt einen String vom Programmspeicher aufs Displays, beginnt an der Cursorposition. Parameter:

– *progmem_s* - Zeiger auf String im Programmspeicher.

Beispiel

Nutzung des alphanumerischen LCD zur Darstellung von Text:

```
#include <homelab/module/lcd_alpha.h>

int main(void)
{
  // Initialisierung des LCD.
  lcd_alpha_init(LCD_ALPHA_DISP_ON);
```

```
  // Löschen des Displays.
  lcd_alpha_clear();

  // Cursor an den Beginn der zweiten Zeile setzen.
  lcd_alpha_goto_xy(0, 1);

  // Anzeige des Texts.
  lcd_alpha_write_string("Hello");
}
```

5.4.9 Graphisches LCD

Related to: *User Interface Module*

Diese Bibliothek enthält die Funktionen zur Nutzung des graphischen LCD des HomeLab. Obwohl das LCD graphische Bilder wiedergeben kann, enthält die Bibliothek diese Funktionen nicht. Das LCD wird als alphanumerisches Display mit 14 x 6 Zeichen genutzt.

Funktionen

void lcd_gfx_init(void)
Intialisiert das LCD.

void lcd_gfx_backlight(bool set)
Schaltet die Hintergrundbeleuchtung des LCD an/aus. Parameter:

- *set* - *true* wenn Hintergrundbeleuchtung an, *false* wenn aus.

void lcd_gfx_clear(void)
Löscht das gesamte Display

void lcd_gfx_clear_line(unsigned char line)
Löscht eine Zeile Text. Parameter:

– *line* - Zeilennummer. 0 bis 5.

void lcd_gfx_goto_char_xy(unsigned char x, unsigned char y)
Wählt eine Position zum Beschreiben aus. Parameter:

– *x* - X-Koordinate. 0 bis 13.

– *y* - Y-Koordinate. 0 bis 5.

void lcd_gfx_write_char(char c)
Schreibt ein Zeichen an eine zuvor bestimmte Position. Parameter:

– *c* - ASCII Zeichen.

void lcd_gfx_write_string(char *s)
Schreibt einen String an eine zuvor bestimmte Position. Parameter:

– *s* - Zeiger auf einen String(char array).

Beispiel

Nutzung des graphischen LCD:

```
#include <homelab/module/lcd_gfx.h>

int main(void)
{
  // Initialisierung des LCD.
  lcd_gfx_init();

  // Löschen des Displays.
  lcd_gfx_clear();

  // Cursor in die Mitte des Bildschirms bewegen.
  lcd_gfx_goto_char_xy(5, 2);

  // Anzeige eines Textes.
  lcd_gfx_write_string("Hello");
}
```

5.4.10 Sensoren

Bezug nehmend auf: *Sensors Module*

Diese Bibliothek enthält Funktionen für die verschiedenen Sensoren im HomeLab kit.

Datentypen

ir_distance_sensor
Infrarotentfernungssensor, Distanzberechnungs-Parameterstruktur. Formel für die Entfernungsberechnung ist *a / (ADC + b) - k*. Struktur:

- *a* - Dividend.
- *b* - Nicht-lineare Konstante.
- *k* - Lineare Konstante.

Konstante

ir_distance_sensor GP2Y0A21YK
Sharp GP2Y0A21YK Formelparameter zur Entfernungsberechnung.

Funktionen

signed short thermistor_calculate_celsius(unsigned short adc_value)
Berechnet die Thermistor Temperatur in Grad Celcius aus ADC Umrechnungsergebnissen. Die Funktionen benutzen eine vorgefertigte Umrechnungstabelle. Parameter:

- *adc_value* - ADC Umrechnungsergebnis (10-Bit mit +5 V Referenzspannung).

– Gibt die Temperatur mit Grenzwerten von -20 bis 100 °C an.

signed short ir_distance_calculate_cm(ir_distance_sensor sensor, unsigned short adc_value)
Berechnet die Entfernung von ADC Ergebnissen die vom IR-Entfernungssensor empfangene Spannung an. Parameter:

– *sensor* - Kalkulationsparameter des Distanzsensors.

– *adc_value* - ADC Umrechnungsergebnis (10-bit mit +5 V Referenzspannung).

– Gibt die Entfernung in cm an, oder -1 wenn eine Messung nicht möglich ist.

unsigned short ultrasonic_measure(pin trigger, pin echo)
Misst die Entfernung mit dem Ultraschallentfernungsmesser. Die Funktionen generieren einen Auslöseimpuls an einem Pin und messen die Zeit bis zum Echopulses am anderen Pin. Die Entfernung wird über die Zeit berechnet. Die Funktionen benötigen eine
14.7456Mhz Taktfrequenz. Die Messung kann bis zu 36ms dauern. Parameter:

– *trigger* - Variable des auslösenden Pins.

– *echo* - Variable des Echo-Pin variable.

– Gibt Entfernung in cm an, oder 0 wenn Messung nicht möglich.

Beispiel

Nutzung des Infrarot- und Ultraschallentfernungssensors.

```
#include <homelab/module/sensors.h>

// Kontroll-Pins des Ultraschallentfernungssensors.
pin pin_trigger = PIN(G, 1);
pin pin_echo    = PIN(G, 0);
```

```
int main(void)
{
  unsigned short adc_value = 400; // zufälliges ADC Ergebnis.
  signed short distance;

  // Entfernungsberechnung des ADC Ergebnisses des IR-
    Entfernungssensors.
  distance = ir_distance_calculate_cm(GP2Y0A21YK, adc_value);

  // Messung mit Ultraschallentfernungssensor.
  distance = ultrasonic_measure(pin_trigger, pin_echo);
}
```

5.4.11 Motoren

Bezug nehmend auf: *Motor Module*

Diese Bibliothek enthält Funktionen zur Steuerung der verschiedenen HomeLab Motoren. Sie enthält Funktionen für Gleichstrom- (DC-), Schritt- und Servomotoren,

Funktionen

void dcmotor_init(unsigned char index)
Initialisiert einen der Controller für die DC Motoren. Parameter:

- *index* - Index des Motorencontrollers. 0 bis 3.

void dcmotor_drive(unsigned char index, signed char direction)
Betriebt einen der Controller der DC Motoren. Parameter:

- *index* - Index des Motorencontrollers. 0 bis 3.
- *direction* - Polarität des Motors. -1, 0 oder +1. Bei 0 wird der Motor angehalten, ansonsten bewegt er sich in die vorgegebene Richtung.

void unipolar_init(unsigned char index)
Initialisiert den Controller eines der unipolaren Schrittmotoren. Parameter:

- *index* - Index des Motorencontrollers. 0 oder 1.

void unipolar_halfstep(unsigned char index, signed char direction, unsigned short num_steps, unsigned char speed)
Befehl zum Halbschrittbetrieb für unipolaren Schrittmotor. Die Funktion ist blockierend bis sie erfüllt ist, bis alle Schritte durchgeführt wurden. Parameter.

- *index* - Index des Motorencontrollers. 0 oder 1.
- *direction* - Rotationsrichtung. -1 oder +1.
- *num_steps* - Zählen der Halbschritte.
- *speed* - Zeit eines einzelnen Schritts in Millisekunden.

void bipolar_init(void)
Initialisiert den Controller des bipolaren Schrittmotors.

void bipolar_halfstep(signed char direction, unsigned short num_steps, unsigned char speed)
Befehl zum Halbschrittbetrieb für bipolaren Schrittmotor. Die Funktion ist blockierend bis sie erfüllt ist, bis alle Schritte durchgeführt wurden. Parameter:

- *direction* - Rotationsrichtung . -1 oder +1.
- *num_steps* - Zählen der Halbschritte.
- *speed* - Zeit eines einzelnen Schritts in Millisekunden.

void servomotor_init(unsigned char index)
Initialisiert eine der PWM Signal-generierenden Einheiten eines Servomotors des ATmega128 Timers 1. Das PWM Signal beträgt 50 hz bei einer high period of 1.5 ms $\pm$ 0.5 ms. Parameter:

- *index* - Index des Servomotors. 0 oder 1.

void servomotor_position(unsigned char index, signed short position)
Takt des Servomotors mit Kontrollbefehl. Die Position eines Servomotors verändert sich wenn dieser betrieben wird. Wird er gedreht, verändert sich seine Drehzahl. Parameter:

- *index* - Index des Servomotors. 0 oder 1.
- *position* - Position oder Drehzahl. -100 bis +100. Bei 0 stoppt der Motor.

Beispiel

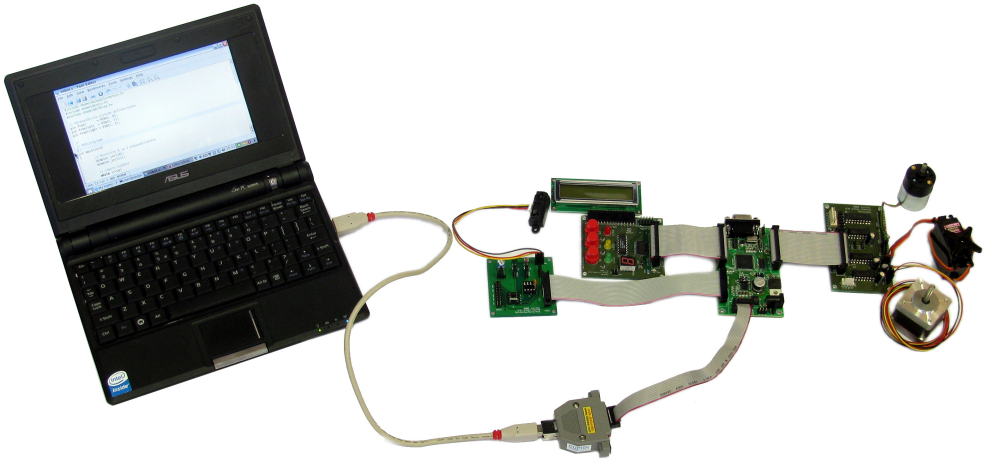
Nutzung von Gleichstrom- (DC-), Schritt- und Servomotoren.

```
#include <homelab/module/motors.h>

int main(void)
{
  // Initialisierung der DC Motoren.
  dcmotor_init(0);
  dcmotor_init(1);

  // Initialisierung des bipolaren Schrittmotors.
  bipolar_init();

  // Initialisierung der Servomotoren.
  servomotor_init(0);
  servomotor_init(1);

  // Ein DC Motor fährt vorwärts, ein anderer rückwärts.
  dcmotor_drive(0, -1);
  dcmotor_drive(1, +1);

  // Bewegt den Schrittmotor 100 Schritte in eine Richtung
  // und daraufhin zurück mit doppelter Geschwindigkeit.
  bipolar_halfstep(1, 100, 50);
  bipolar_halfstep(-1, 100, 25);

  // Bewegt Servomotoren in entgegengesetzte Richtungen.
  servomotor_position(0, -100);
  servomotor_position(1, +100);
```

```
}
```

— MISSING PAGE —

6 Praktische Beispiele



Die, in diesem Kapitel vorgestellten, praktischen Beispiele sind einheitlich aufgebaut und so konkret und praxisnah wie möglich gestellt. Jedes Beispiel beginnt mit einer kurzen Einführung in die zugrunde liegende Theorie sowie einer Darstellung des zum Verständnis der Beispiele nötigen Hintergrundwissens. Der praktische Teil besteht hauptsächlich aus kommentierten Programmcode, welcher die konkrete Anwendung verdeutlicht. Normalerweise wird dabei die HomeLab Library genutzt, in einigen Fällen wird die Hardware jedoch auch direkt über Register angesteuert.

Das erste Kapitel weicht etwas von diesem Schema ab, hier wird primär die Installation und Konfiguration der notwendigen Softwarekomponenten beschrieben. Die Installationsanweisungen sind für Windows und Linux Betriebssysteme ausgelegt. Nachdem die Software einmal eingerichtet wurde, macht es keinen Unterschied mehr, ob ein Windows oder ein Linux/Unix basierendes Betriebs-

system verwendet. Die Handhabung und Programierung wird dadurch nicht beeinflusst.

Ein praktisches Beispiel beginnt immer mit einer Aufzählung von erforderlichen Vorkenntnissen. Dabei werden Bezüge zu anderen Kapiteln hergestellt, welche mit folgenden Abkürzungen gekennzeichnet sind:

Physikalisches Modul, das im Beispiel genutzt wird.

Teil der Softwarebibliothek, die im Beispiel genutzt wird.

Verweis auf das Kapitel über die Grundlagen der Elektronik.

Verweis auf das Kapitel über die AVR Mikrocontrollermodule.

Verweis auf andere Aufgaben.

Grundlegende Software für die Beispiele

Wie zuvor erwähnt werden die Codebeispiele auf Basis der HomeLab Bibliothek erstellt.Die meisten AVR-spezifischen Operationen und Verfahren, die auf die HomeLab Kit Hardware zugreifen sind in dieser Bibliothek enthalten. Bei Verwendung der HomeLab Bibliothek muss somit kein Hardware-spezifischer Programmcode mehr erzeugt werden. Durch Bereitstellung der für die Hardware in den Beispielen und Übungen notwendigen Software,kann der Nutzer sich vollkommen auf seine Algorithmen konzentrieren und muss keine Zeit mit Einzelheiten verbringen, die nicht für die zu benutzende Hardware notwendig sind.

Programmierstil in den Beispielen

Die Beispielprogramme sind in einem einheitlichen Stil geschrieben, um so einen synoptischeren Programmcode zu erhalten. So wird das Programm leserlicher und einfache Syntaxfehler können vermieden werden. Daher sollte dieser Stil auch bei den Übungsaufgaben angewandt werden. Die Hauptcharakteristika sind:

Das Programm, all seine Funktionen und Variablen sind in englischer Sprache und Kleinbuchstaben geschrieben, die Wörter werden durch einen Unterstrich getrennt.

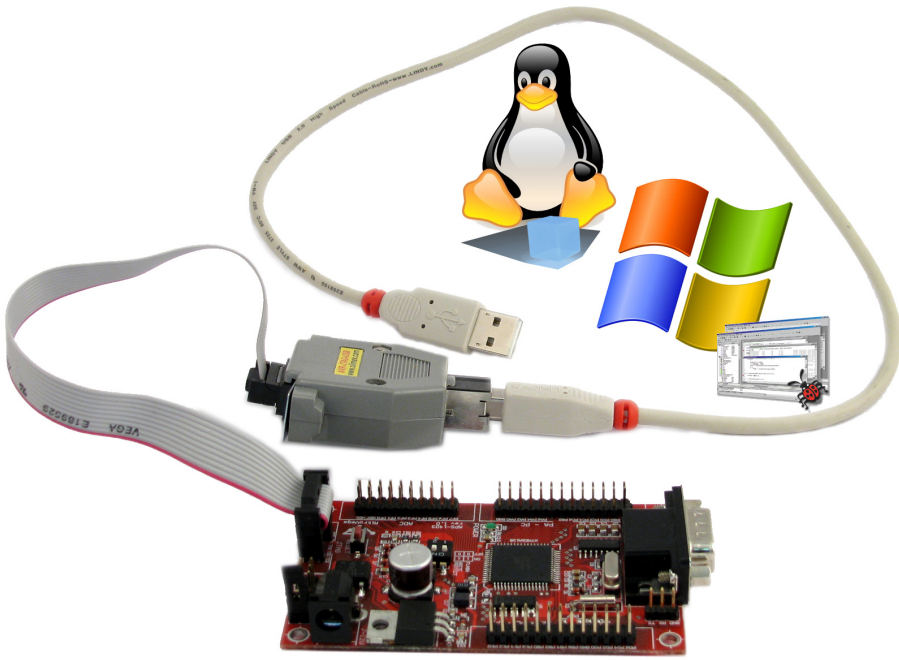
Die Funktionen werden wie folgt gekennzeichnet: *object_action_subject*.

Wichtigere Teile des Programms werden kommentiert.

Jeder Block in C-Code (markiert durch und)startet und beginnt in einer neuen Zeile.

Blöcke werden mit der Tabulatortaste geordnet. Die Tabulatortaste wird nicht innerhalb einer Zeile verwendet.

6.1 Anfang



Zur Programmierung des AVR Mikrocontrollers werden eine Programmierumgebung, ein Compiler für die betreffende Sprache sowie Software zum Laden der erstellten Programme auf den Controller benötigt. Die komfortabelste Weg hierfür ist die Nutzung der speziellen DIE (Integrated Developing Environment). Der AVR Mikrocontroller kann in folgenden Programmiersprachen programmiert werden: Assembler, C, C+, Pascal, Basic, etc. In diesem Buch wird C verwendet um den Mikrocontroller zu programmieren. Die Software ist sowohl für Windows OS und Linux OS frei erhältlich. Die folgenden Kapitel führen in die Verwendung dieser ein.

Sämtliche, zur Durchführung der Beispiele notwendige Software, ist auf der HomeLab-Homepage zu finden.

6.1.1 Installationsanweisung für Windows

Dieser Abschnitt enthält alle notwendigen Informationen zur Installation der AVR Entwicklungssoftware auf einem Windows Betriebssystem zu installieren.

Installation

Es wird die nachfolgende Software benötigt, die von der Homepage des Herstellers heruntergeladen werden kann. Wahlweise kann die Software auch von der, im HomeLab Kit enthaltenen, CD installiert werden.

1. AVR Studio

AVR Studio 4 IDE (*Integrated Development Environment*) ist eine Software um den Quellcode zu schreiben, zu kompilieren und das Programm in den Controller zu laden. AVR Studio kann kostenlos von der Atmel Homepage bezogen werden.

2. WinAVR

WinAVR ist ein GNU-GCC Compiler für AVR Microcontroller. Das Programm ist eine kostelose Software und kann welche von der Sourceforge Homepage gela-

den werden. Währen der Installation von WinAVR schlägt das Programm einen Ordnernamen mit einer langen Versionsnummer vor. Wir würden vorschlagen, die Versionsnummer zu entfernen, und das Programm wie folgt zu installieren:

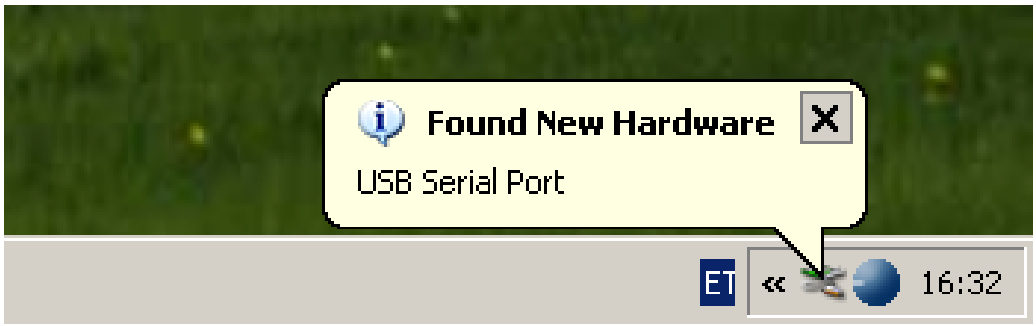
```
C:\WinAVR
```

3. HomeLab Bibliothek

Die HomeLab Bibliothek ist ein Set von Funktionen welches für das HomeLab Kit und AVR Controller entworfen wurde. Die Nutzung dieser Bibliothek macht das Programmieren wesentlich einfacher und effektiver. Die neuste Version der Bibliothek kann von der HomeLab Webseite heruntergeladen werden. Diese Bibliothek sollte in den gleichen Ordner installiert werden, in dem sich auch WinAVR befindet.

4. Virtualer COM Port Treiber

Der Treiber ermöglicht den Anschluss des USB JTAG-ICE Programmiergerätes an den PC. Er muss vor Anschluss des Programmiergerätes installiert werden. Der Dateiname des Treibers ist „CDM x.xx.xx.exe“, wobei „x“für die Versionsnummer steht. Nach der Installation des Treibers kann das JTAG-ICE Programmiergerät über den USB Port angeschlossen werden und Windows wird es automatisch erkennen. Folgende Meldung sollte in der Taskleiste zu sehen sein:



Abhängig von der Anzahl vorher bereits definierter virtueller Ports, wird Windows dem neuen Gerät eine Portnummer zuweisen. Es wird ein neuer Name für den COM Port generiert, sobald das Gerät an einen anderen USB Port des selben Computers angeschlossen wird. Einige AVR Studio Versionen können nicht auf

das Programmiergerät zugreifen, wenn die Nummer des COM Ports größer ist als 10, bzw. größer als 4 wenn es als Debugger genutzt wird.

Um das zu verhindern, kann die Portnummer im *Gerätemanager* verändert und auf einen Wert zwischen COM0 und COM4 einstellt werden.

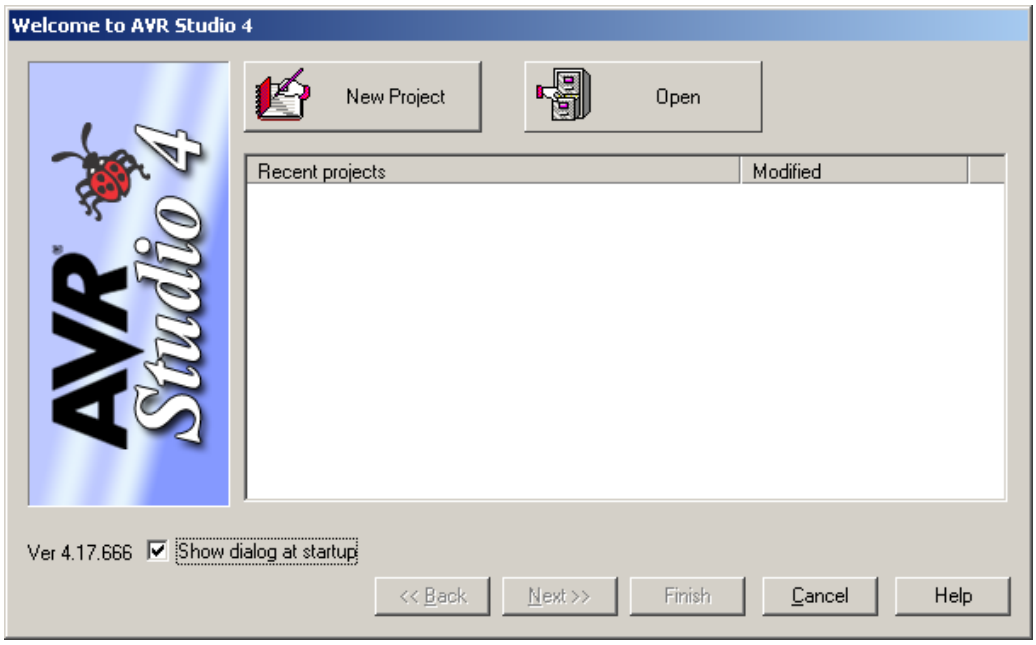
See the complete procedure here .

Ein neues Projekt erstellen

Um ein Programm für den Controller zu schreiben muss man einen Projektraum erstellen. Das Projekt beinhaltet typischerweise unterschiedliche Dateien wie Quellcodes, Headerdateien, kompilierte Programmdateien usw. Es wird dringend dazu geraten, für jedes Projekt einen separaten Ordner zu erstellen (Diese Funktion wird auch vom New Project Wizard angeboten).

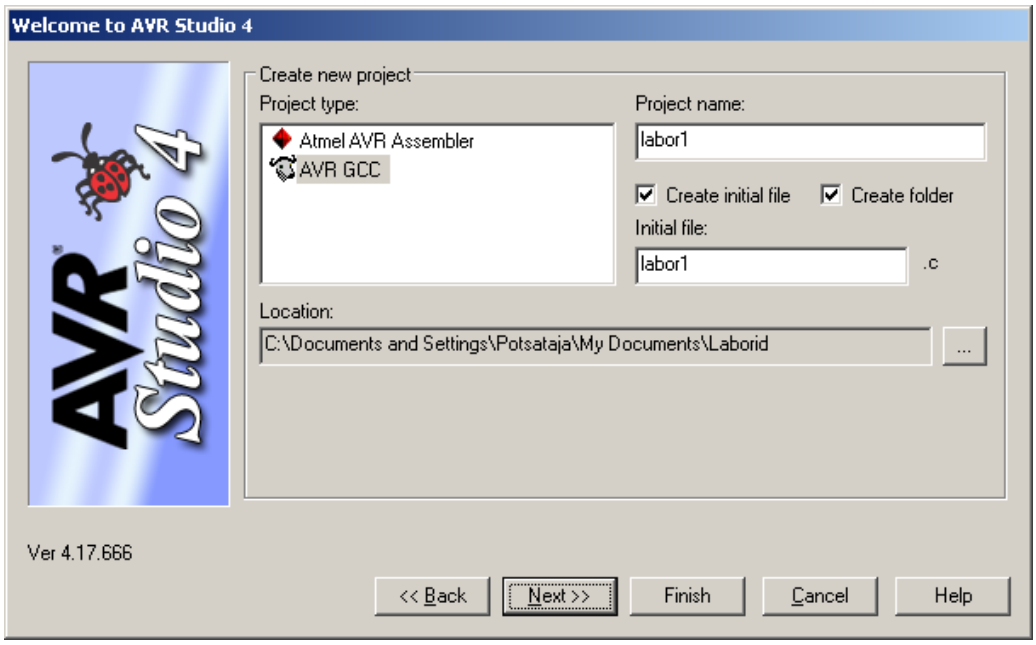
Wird ein neues Projekt mit Hilfe des Wizards erstellt, sind folgende Schritte zu durchlaufen:

1. Öffnen Sie AVR Studio und klicken Sie auf *New Project*. Falls sich das Dialogfeld nicht automatisch öffnet, wählen Sie *Project - New project* aus der Menüleiste aus. Klicken Sie auf *Next*.

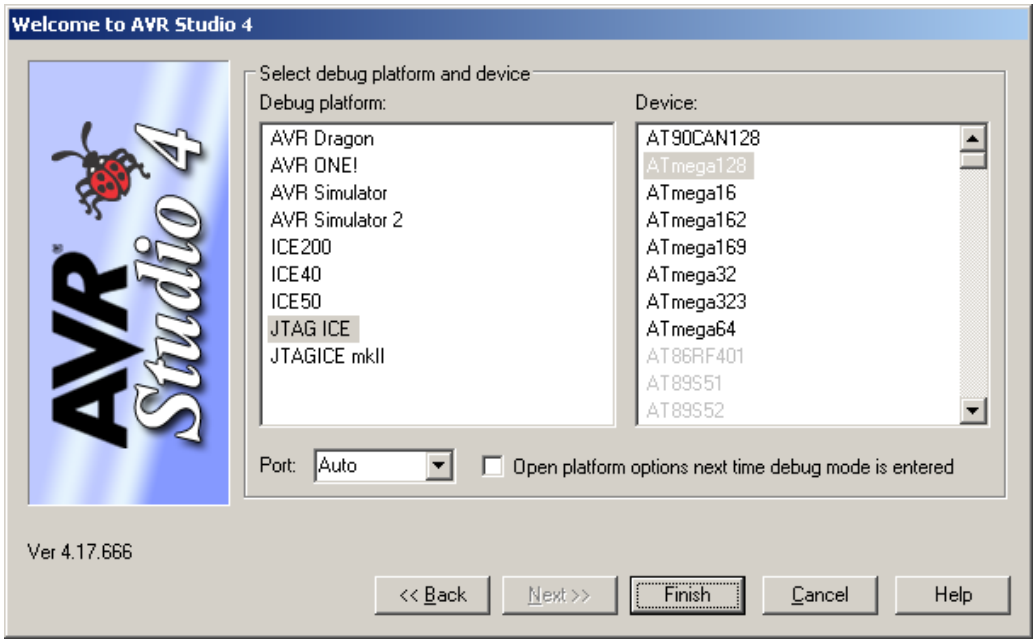


2. In dem nächsten Dialogfeld geht es um den Compiler und die Starteinstellungen. Wählen Sie den AVR GCC Compiler aus. Fügen Sie dann links den Namen des Projekts und den Hauptquelldateinamen ein. Der Quelldateienname sollte die Endung „*.c“haben. Außerdem sollten noch zwei Ankreuzfelder angeklickt werden, welche einen Neuen Ordner und die Startdatei erzeugen. Man sollte auch den Ordner angeben wo die Projektdateien erstellt werden. Nachdem diese Einstellungen vorgenommen wurden, klicken Sie auf *Next*.

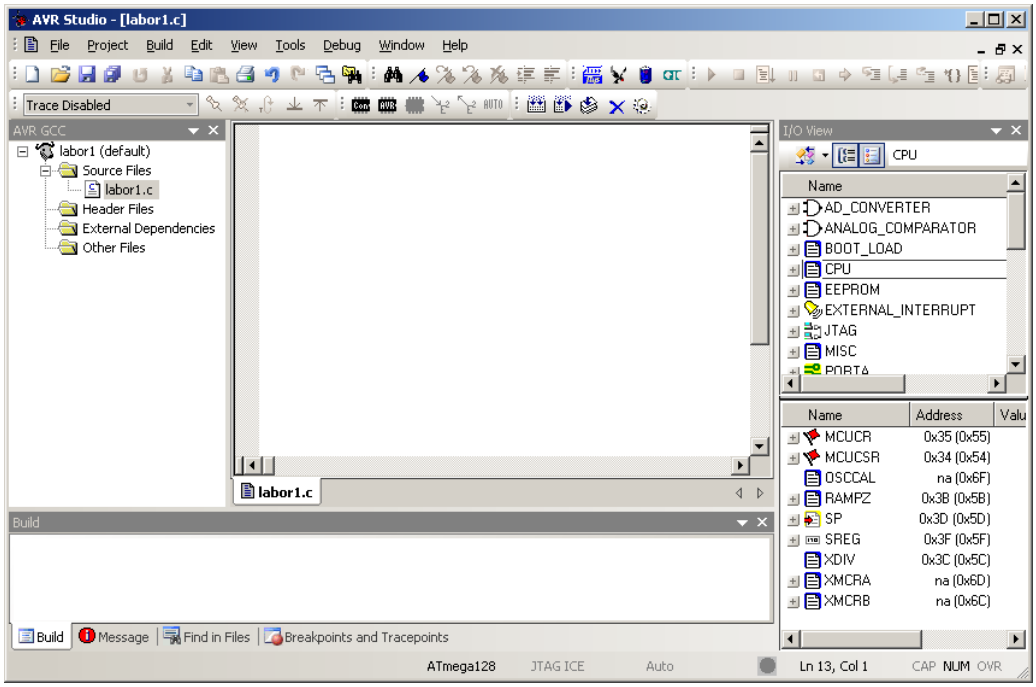
Achtung! Fehlt der AVR GCC in der Compilerliste, ist er nicht richtig installiert. In diesem Fall muss die WinAVR Software installiert werden bevor mit dem Programmieren begonnen wird.



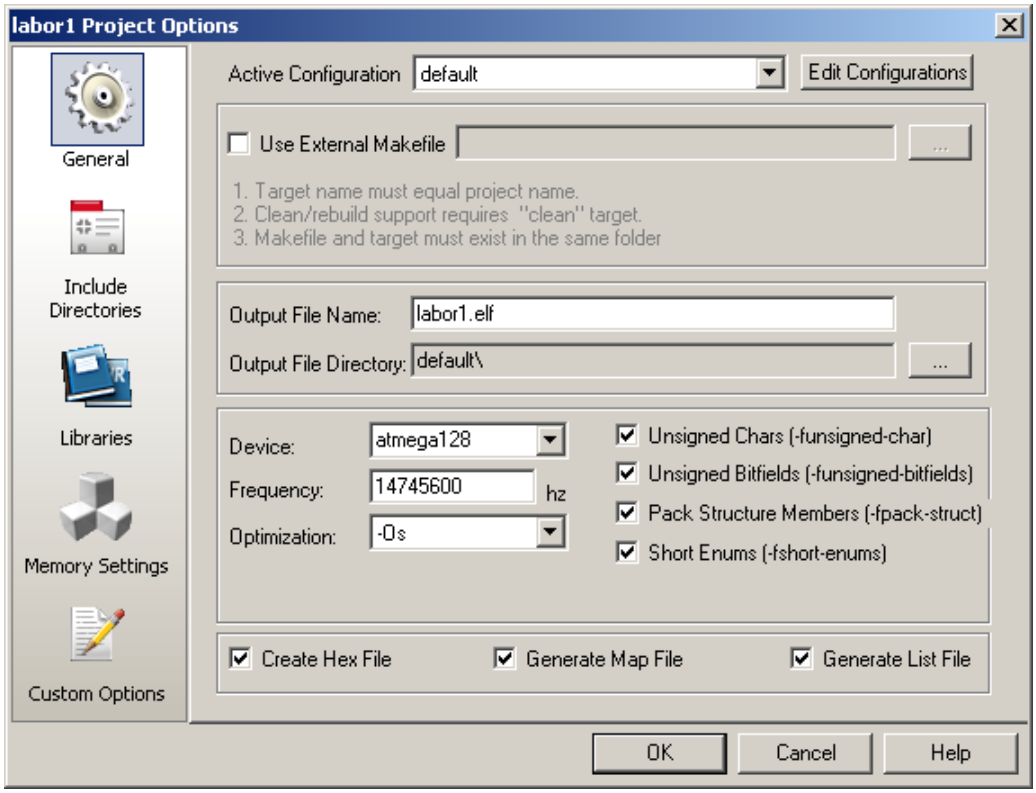
3. Im nun folgenden Dialogfeld sind die Debuggerplattform sowie der Mikrocontroller auszuwählen. Das HomeLab kit nutzt die JTAG ICE Debuggerplattform und den ATmega128 Mikrocontroller. Klicken Sie nach der Auswahl auf *Finish*.



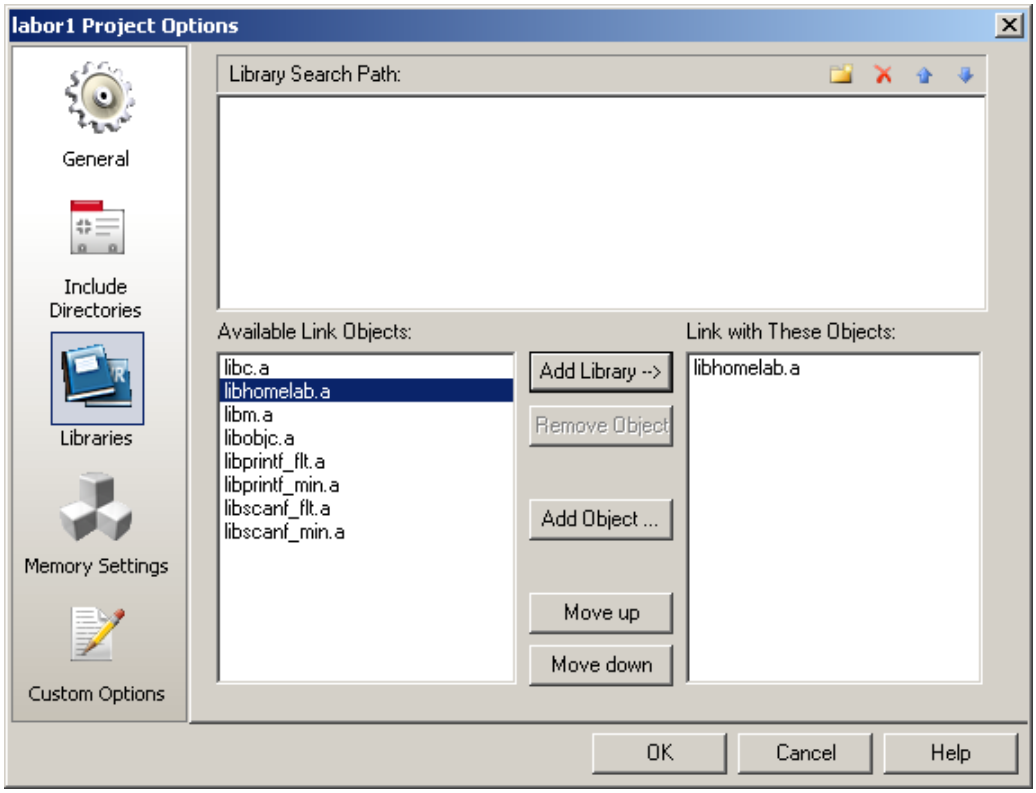
4. Nun wird der Projektraum erstellt und ein neues Fenster öffnet sich, in welchem mit dem Programmieren begonnen werden kann.



5. Vor der ersten Kompilierung, müssen einige Parameter des Projekts eingestellt werden.Die wichtigen Parameter sind die Taktfrequenz des Controllers und Optimierungsmethode.Der HomeLab Controller hat eine Frequenz von 14,7456 MHz (14745600 Hz). Sie kann in den Projekteinstellungen in Hz (nicht MHz) angegeben werden : *Project* → *Configuration Options* → *General*. Die Optimierungsmethode sollte -Os sein, falls keine andere Methode benötigt wird.



6. Um die Funktionen der HomeLab Bibliothek zu nutzen muss die Software richtig installiert sein. Bei jedem neuen Projekt benötigt muss die Bibliothek zur Liste der verlinkten Objekte hinzugefügt werden. Hierzu gehen Sie auf: *Project* → *Configuration Options* → *Libraries* und fügen Sie „libhomelab.a“hinzu.



Falls das Objekt *libhomelab.a* in der Liste links fehlt, ist die Bibliothek nicht richtig installiert und sollte erneut installiert werden.

Test der Einstellungen

Nach Installation der Entwicklungsumgebung sollte diese getestet werden, um sicherzustellen, dass sie korrekt funktioniert. Der einfachste Weg hierfür ist, ein kleines Programm zu schreiben, es zu kompilieren und in den Controller zu laden.

1. Schließen Sie das Programmiergerät an dem ATmega128 an. Stellen Sie sicher, dass es korrekt am JTAG Anschluss angeschlossen ist. (Kabel zeigt vom Board weg - siehe Abbildung) Schließen Sie nun die Stromversorgung an (eine kleine grüne LED sollte leuchten wenn alles korrekt angeschlossen ist.).

Fügen Sie nun folgenden C Code ein:

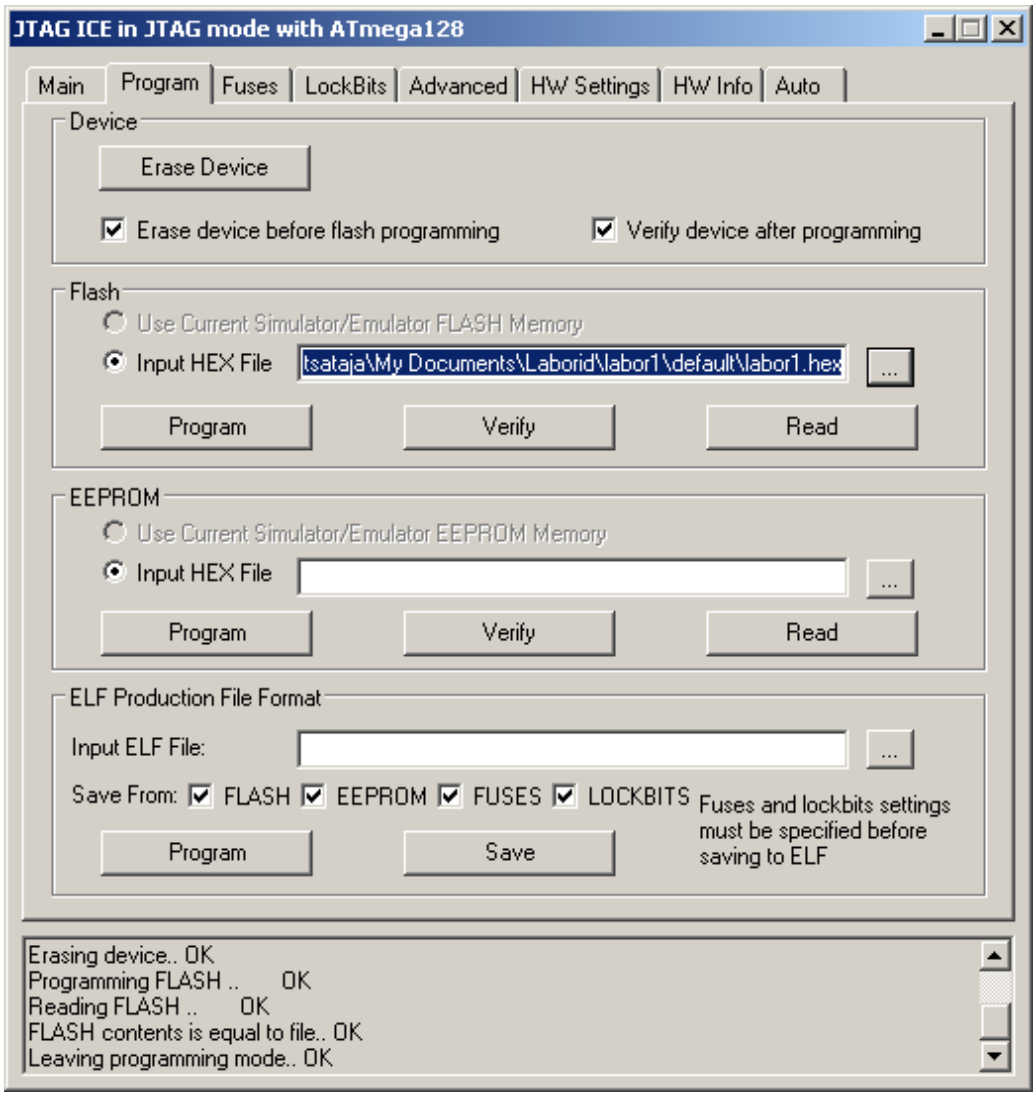
```
#include <avr/io.h>
#include <homelab/delay.h>

int main(void)
{
  // Pin PB7 als Output
  DDRB = 0x80;

  // Endlosschleife
  while (true)
  {
    // Invertierung Pin PB7
    PORTB ^= 0x80;
    hw_delay_ms(500);
  }
}
```

Abbildung 6.1: Kompilierung und Schalter des Programmiergerätes

Kompilieren Sie das Projekt mit dem *Build* Befehl (Taste F7). Stellen Sie sicher, dass die Kompilierung erfolgreich ist. Dafür sollte folgende Nachricht im Nachrichtenfenster stehen:

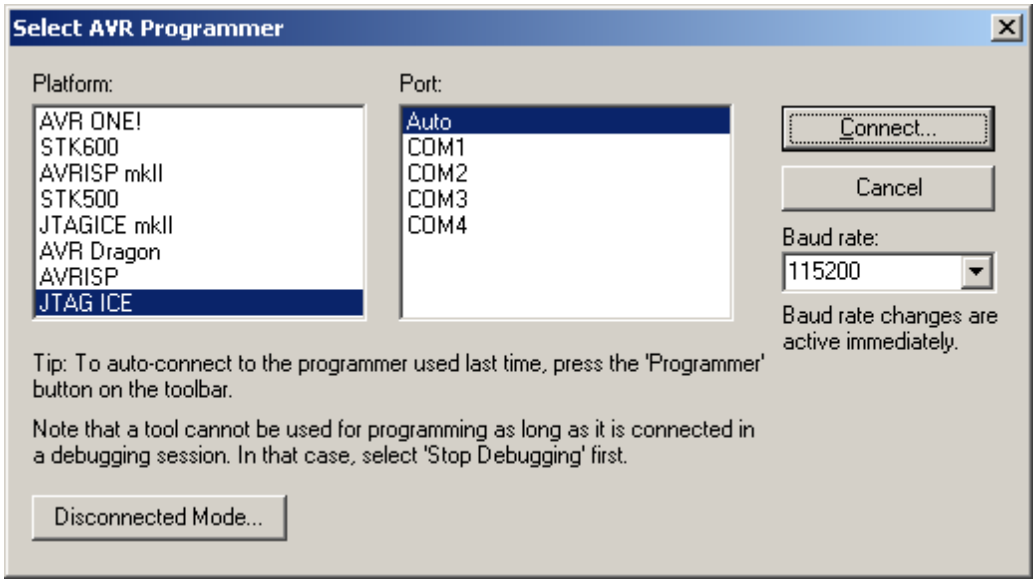
```
Build succeeded with 0 Warnings...
```

2. Öffnen Sie das Controllerfenster *Tools* → *Program AVR* → *Auto Connect*. Stellen Sie sicher, dass der Tab *Program* geöffnet ist.



Falls sich das beschriebene Fenster nicht öffnet und sich stattdessen das *Connection Failed* Fenster öffnet, ist das Programmiergerät oder die Platine nicht richtig angeschlossen. Überprüfen Sie zunächst, ob der Mkcrocontroller an die Stromversorgung angeschlossen ist (grüne LED leuchtet) und ob das Programmiergerät korrekt an den JTAG angeschlossen ist. Falls das korrekt erfolgt ist, überprüfen Sie die von Windows vergebene COM Port Nummer. Ist diese größer als 9, kann Das AVR Studio das Programmiergerät nicht erkennen. Folgen Sie den

Anweisungen zu Beginn des Kapitels, um eine Portnummer zwischen 0 und 4 zuzuweisen.

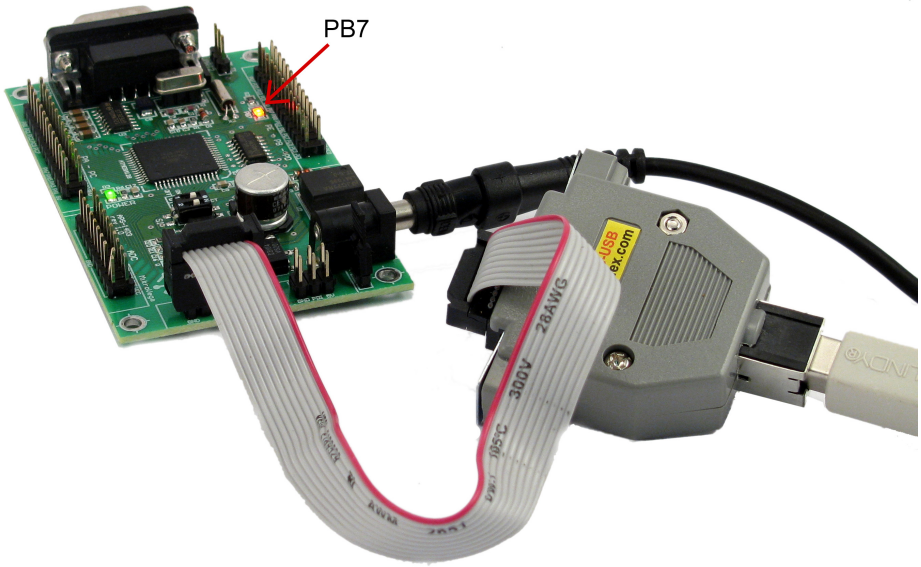


3. Fügen Sie im Fenster des Programmiergerätes *Flash*-sektion Textbox *Input HEX File* durch Betätigung der "..."Taste den Ort des kompilierten Programms ein. Das kompilierte Programm ist normalerweise in dem Unterordner *default* des Projekts zu finden und hat den gleichen Namen wie das Projekt jedoch mit der Endung „*.hex", z.B. „Labor1.hex". Nach Auswahl der richtigen Datei wählen Sie *Programm* um das Programm in den Controller zu laden.Wenn alles richtig gelaufen ist, sollte folgende Nachricht am Ende des Fensters zu sehen sein:

```
OK
Reading FLASH input file.. OK
Setting device parameters for jtag programming ..OK
Entering programming mode.. OK
Erasing device.. OK
Programming FLASH ..        OK
Reading FLASH ..        OK
FLASH contents is equal to file.. OK
Leaving programming mode.. OK
```

Laut Programm sollte die on-Board LED (PB7) blinken. Wenn das Programm funktioniert ist das Setup der Programmierumgebung richtig und das erste Pro-

gramme wurde erfolgreich kompiliert.Glückwunsch!



Debugger

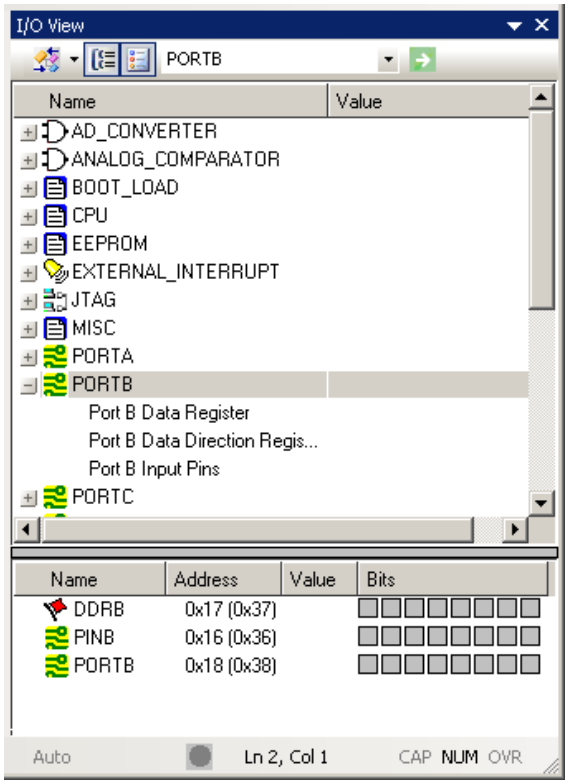


Abbildung 6.2: Liste der Register des Debuggers des ATmega128.

Debuggen eines Programms bedeutet, dass man nach Fehlern im Programm sucht.Dazu gibt es Programme, genannt Debugger, welche erlauben das Programm Schritt-für-Schritt auszuführen und an den notwendigen Stellen zu stoppen. Eine Implementierung des Programms ermöglicht das Überprüfen von Werten einer Variable in jeder Phase des Programms, sowie des Inhalts der Register und der Sequenz des Ausführens des Programms.Debugging ist sehr wichtig, wenn man mit komplexeren Programmen arbeitet, in denen die Fehlersuche häufig schwierig ist. Bei der Arbeit mit Mikrocontrollern ist es wichtig, dass eine Schritt-für-Schritt Implementation des Programms im Controller stattfindet, wodurch die Veränderung des tatsächlichen Outputs und auch die Werte der Register gesehen werden können. Zur Nutzung eines Debuggers müssen zwei erfüllt sein: Der Mikrocontroller muss das Debugging unterstützen und man muss über

die notwendige Hardware verfügen, nämlich den JTAG Programmer welcher Debuggen erlaubt. Günstigere Programmiergeräte, welche das ISP Programmier Interface nutzen, können zwar kompilierte Programme in den Controller laden aber nicht unbedingt debuggen.

Um das Programm im Debugmodus mit dem AVR Studio zu starten, sollte es zuerst durch *build* (F7) kompiliert und daraufhin mit dem Befehl *run* (F5) gestartet werden.Zuvor können *break points* (F9) an bestimmten Stellen im Quellcode eingefügt werden.Erreicht die Implementation des Programms einen solchen Punkt, wird das Programm angehalten, um den Status des Mikrocontrollers an diesem Punkt festzulegen. Die Implementation des Programms kann mit dem Befehl *Run* fortgeführt oder durch *Step Into* (F11) mit nur einem Befehlt vollständig implementiert werden.

Die Nutzung von floating-point Variablen

Manchmal ist es bei AVR Programmen notwendig, Floating-Point Variablen zu nutzen. Um mit ihnen zu rechnen und sie mit *printf*-Typ Funktionen darzustellen, müssen folgende Installationsänderungen in der Konfiguration des Projekts vorgenommen werden:

1. Öffnen Sie die Installationsdatei des Projekts aus dem Menü *Project* → *Configuration Options*. Fügen Sie *libprintf_flt.a* und *libm.a* zu *libhomelab.a* der Homelab Bibliothek im Konfigurationstab *Libraries* hinzu.

2. Öffnen Sie anschließend *Custom Options* und wählen Sie *[All files]*. Fügen Sie daraufhin Zeilen mit "-lprintf_flt"ja "-lm"in die Box rechts sowie die Zeile mit "-uvfprintf"zu der *[Linker Options]* Sektion hinzu.

3. Klicken Sie auf OK und schließen Sie das Konfigurationsfenster.

6.1.2 Die Softwareumgebung für Linux

Die folgende Anleitung beschreibt die Installation und de Benutzung des AVR Toolkit 9:10 Ubuntu OS.

Installation der Software

Installieren Sie die folgende Software:

1. Linux Softwarepakete

gcc-avr - GNU C-Compiler für den AVR

avrdude - Programm um die HEX-Dateien in den Controller zu laden.

avr-libc - C-Code Bibliotheken, um Code für die AVR Plattform zu schreiben

Um die Pakete zu installieren, nutzen Sie folgenden Terminalbefehl:

```
sudo apt-get install gcc-avr avrdude avr-libc
```

oder die graphische Installationssoftware (wie das Ubuntu Softwarecenter oder Synaptic Packet manager).

2. HomeLab Bibliothek

Die Bibliothek vereinfacht das schreiben von Programmcode, da Grundfunktionen bereits fertig sind. Laden Sie sich die Datei *Homelab library vX.X.run* von der HomeLab Webseite herunter, wobei XX für die Versionsnummer steht. Zur Installation der Bibliothek führen Sie den folgenden Befehl aus:

```
sudo sh homelab_library_vX.X.run
```

Stellen Sie sicher, dass die Installation erfolgreich war.

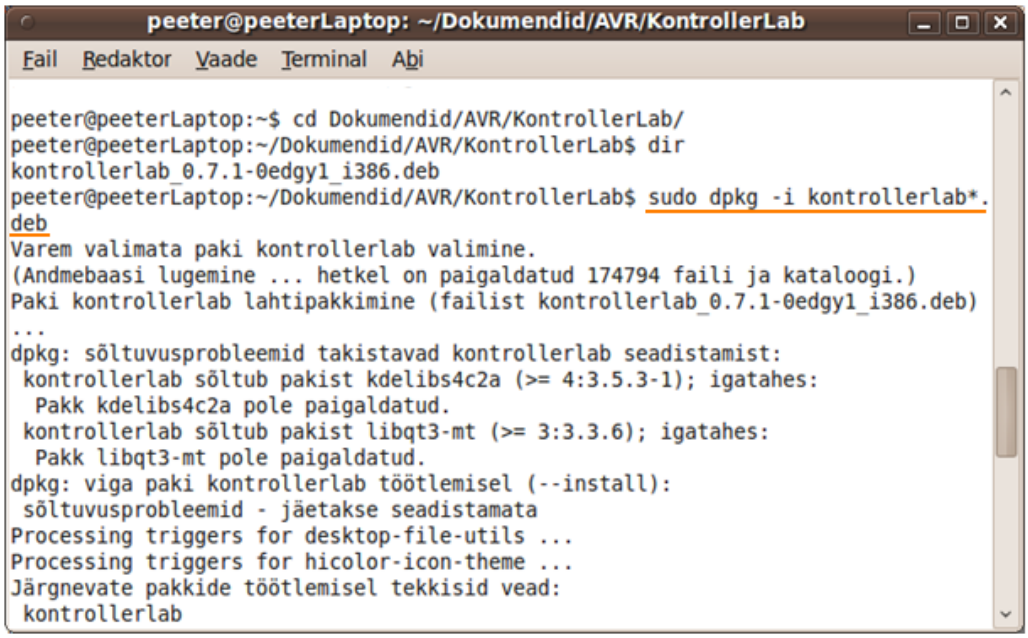
3. KontrollerLab

KontrollerLab ist eine IDE (*integrated development environment*) um zum schreiben, kompilieren, debuggen und laden von AVR Software. Laden Sie die KontrollerLab Software herunter und und speichern sie diese in einem Ordner (z.B. *~/Documents/AVR/KontrollerLab*). Führen sie den folgenden Befehl in dem Ordner aus:

```
sudo dpkg -i kontrollerlab*.deb
```

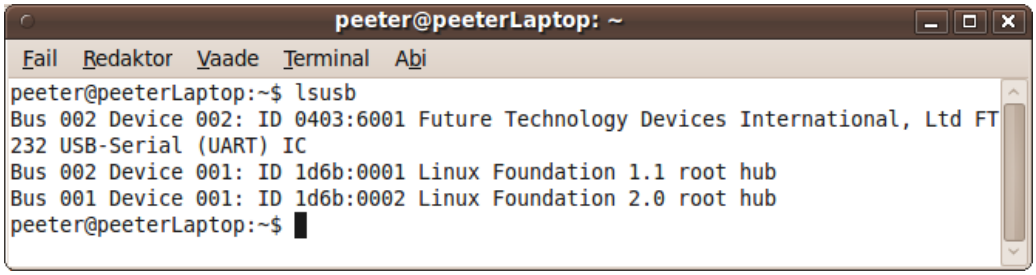
Falls es Probleme mit Paketabhängigkeiten gibt, führen Sie den Befehl aus welcher die fehlende Pakete installiert:

```
apt-get install -f
```

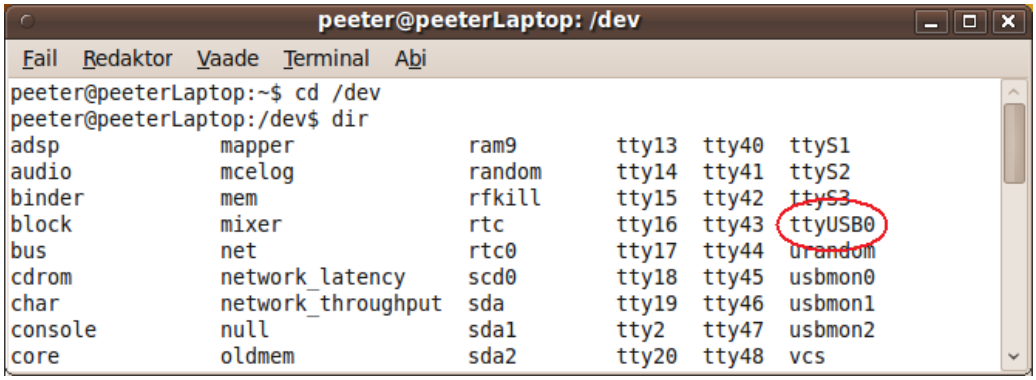

peeter@peeterLaptop: ~/Dokumendid/AVR/KontrollerLab
Fail Redaktor Vaade Terminal Abi
peeter@peeterLaptop:~$ cd Dokumendid/AVR/KontrollerLab/
peeter@peeterLaptop:~/Dokumendid/AVR/KontrollerLab$ dir
kontrollerlab_0.7.1-0edgy1_i386.deb
peeter@peeterLaptop:~/Dokumendid/AVR/KontrollerLab$ sudo dpkg -i kontrollerlab*.deb
Varem valimata paki kontrollerlab valimine.
(Andmebaasi lugemine ... hetkel on paigaldatud 174794 faili ja kataloogi.)
Paki kontrollerlab lahtipakkimine (failist kontrollerlab_0.7.1-0edgy1_i386.deb)
...
dpkg: sõltuvusprobleemid takistavad kontrollerlab seadistamist:
kontrollerlab sõltub pakist kdelibs4c2a (>= 4:3.5.3-1); igatahes:
Pakk kdelibs4c2a pole paigaldatud.
kontrollerlab sõltub pakist libqt3-mt (>= 3:3.3.6); igatahes:
Pakk libqt3-mt pole paigaldatud.
dpkg: viga paki kontrollerlab töötlemisel (--install):
sõltuvusprobleemid - jäetakse seadistamata
Processing triggers for desktop-file-utils ...
Processing triggers for hicolor-icon-theme ...
Järgnevate pakkide töötlemisel tekkisid vead:
kontrollerlab

Das Programmiergerät anschließen.

Schließen Sie das Programmiergerät an den Computer an und überprüfen Sie ob es vom Computer erkannt wird. Schreiben Sie den Befehl „lsusb“ins Terminalfester, woraufhin eine Liste der angeschlossenen USB Geräte ausgegeben werden sollte. Der Name des Programmiergerätes ist: „Future Technology Devices International, Ltd FT 232 USB-Serial (UART) IC“.


peeter@peeterLaptop: ~
Fail Redaktor Vaade Terminal Abi
peeter@peeterLaptop:~$ lsusb
Bus 002 Device 002: ID 0403:6001 Future Technology Devices International, Ltd FT
232 USB-Serial (UART) IC
Bus 002 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
peeter@peeterLaptop:~$

Um den Port zu erhalten an welchem das Programmiergerät angeschlossen ist, überprüfen Sie */dev* mit dem Befehl *cd /dev* (setzt */dev* als aktuellen Ordner) und *dir* (zeigt den Inhalt des Ordners an). Da das Programmiergerät eine serielle USB Schnittstelle ist, ist es als „ttyUSBx"benannt, wobei „x"den Index der Schnittstelle anzeigt.Gibt es kein anderes serielles USB Gerät, ist diese Zahl null.

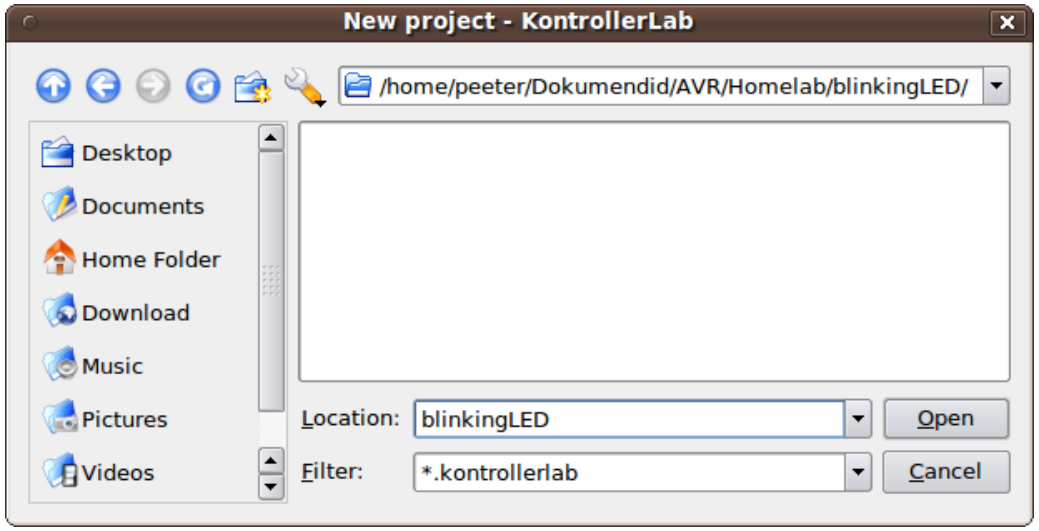

peeter@peeterLaptop: /dev
Fail Redaktor Vaade Terminal Abi
peeter@peeterLaptop:~$ cd /dev
peeter@peeterLaptop:/dev$ dir
adsp mapper ram9 tty13 tty40 ttyS1
audio mcelog random tty14 tty41 ttyS2
binder mem rfkill tty15 tty42 ttyS3
block mixer rtc tty16 tty43 ttyUSB0
bus net rtc0 tty17 tty44 urandom
cdrom network_latency scd0 tty18 tty45 usbmon0
char network_throughput sda tty19 tty46 usbmon1
console null sda1 tty2 tty47 usbmon2
core oldmem sda2 tty20 tty48 vcs

Ein neues Projekt erstellen

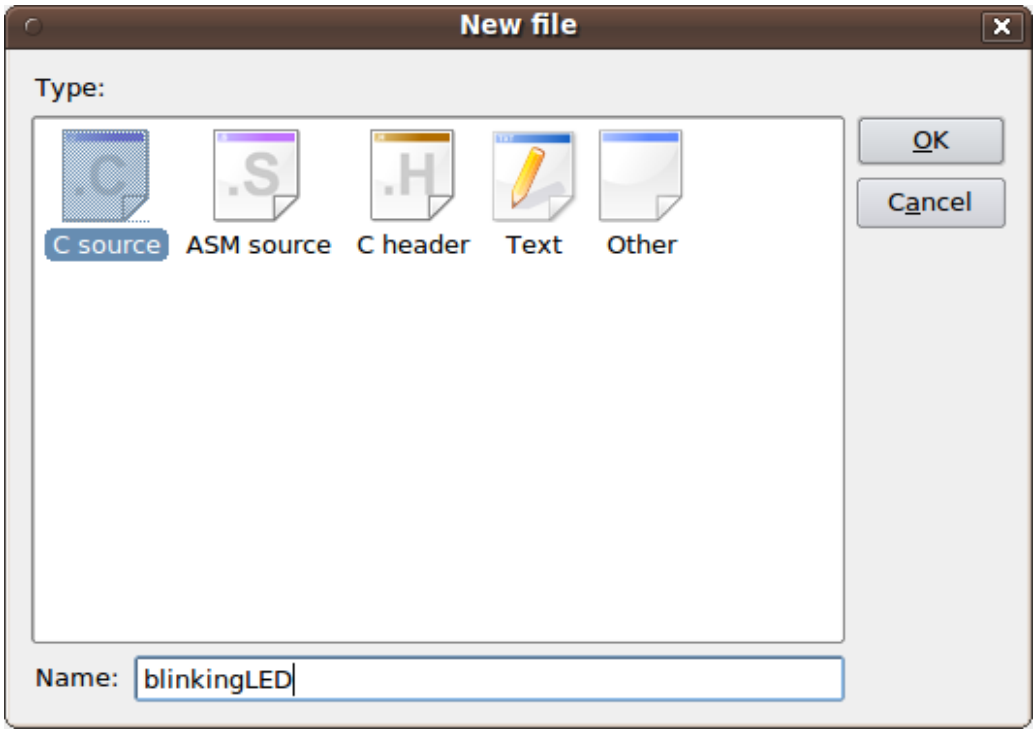
Um ein AVR Programm zu schreiben, muss ein Projekt erstellt werden, welches alle notwendigen Dateien enthält: Quellcode, Headerdateien, kompilierte Programme, usw. Um die Projekte klar voneinander zu trennen, sollte für jedes Projekt ein eigener Ordner erstellt werden. (Diese Option gibt es auch im Project Wizard).

Folgende Schritte müssen durchlaufen werden um ein Projekt zu erstellen:

1. Öffnen Sie KontrollerLab (*Applications* → *Programming* → *KontrollerLab*) und starten Sie ein neues Projekt *File* → *New* → *New project*. Ein Fenster öffnet sich in welchem der Projektort festgelegt werden muss. In diesem Beispiel ist das *../Homelab/blinkingLED/*.

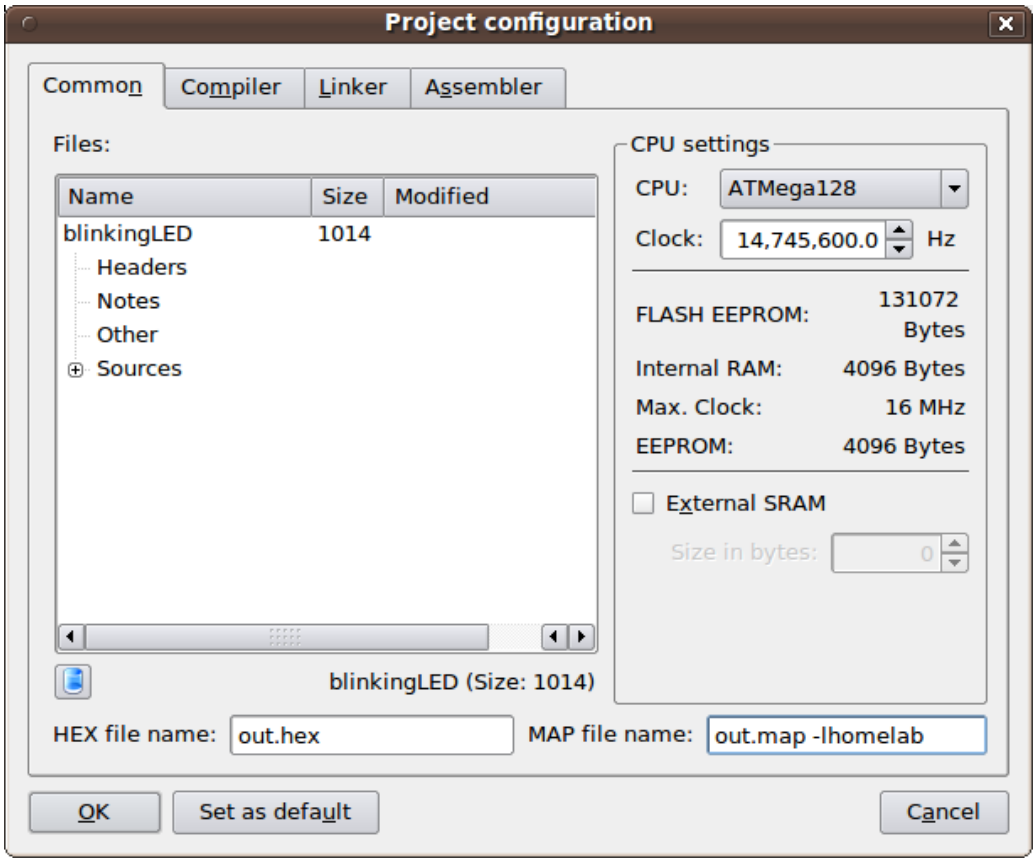


2. Das Projekt muss mindestens eine C-Datei haben in welcher der Programmcode geschrieben wird. Dateien können mit dem *File* → *New* → *New* Menu hinzugefügt werden. In dem sich nun öffnenden Fenster wählen Sie *C source* als Dateityp und wählen Sie einen Namen für die Datei.

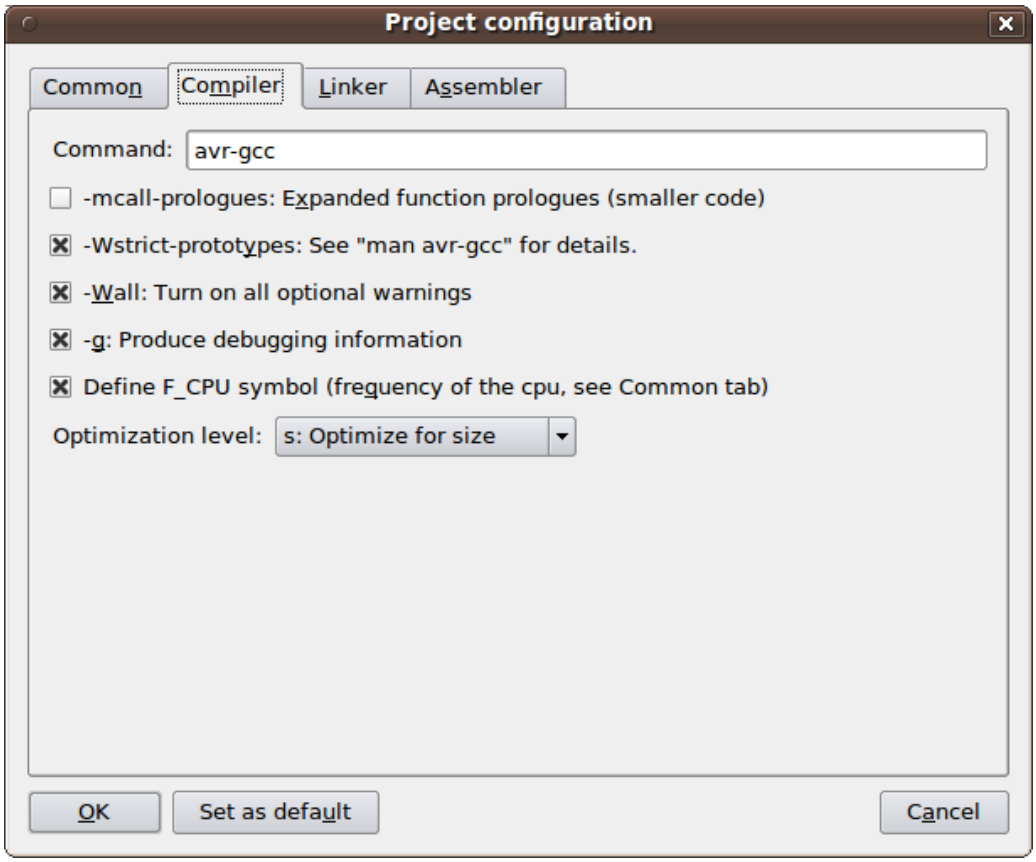


3. Anschließend muss das Projekt für das Homelab Kit konfiguriert werden. Öffnen Sie hierzu das Konfigurationsfenster unter *Project* → *Configure Project* und wählen Sie *Common*. Der CPU Type sollte ATmega128 sein und die Taktfrequenz 14745600,0 Hz. Darüber hinaus sollten hier die Namen der HEX- und MAP- Dateien angegeben werden. Durch *Set as default* werden diese Einstellungen zur Standardeinstellung für alle neuen Projekte. Dieses ist sinnvoll, wenn nur mit dem HomeLab AVR Microcontroller gearbeitet wird. Da die HEX und MAP Dateien als Standard gespeichert werden, sollte man ihnen allgemeine Namen geben, z.B. „out.hex“.

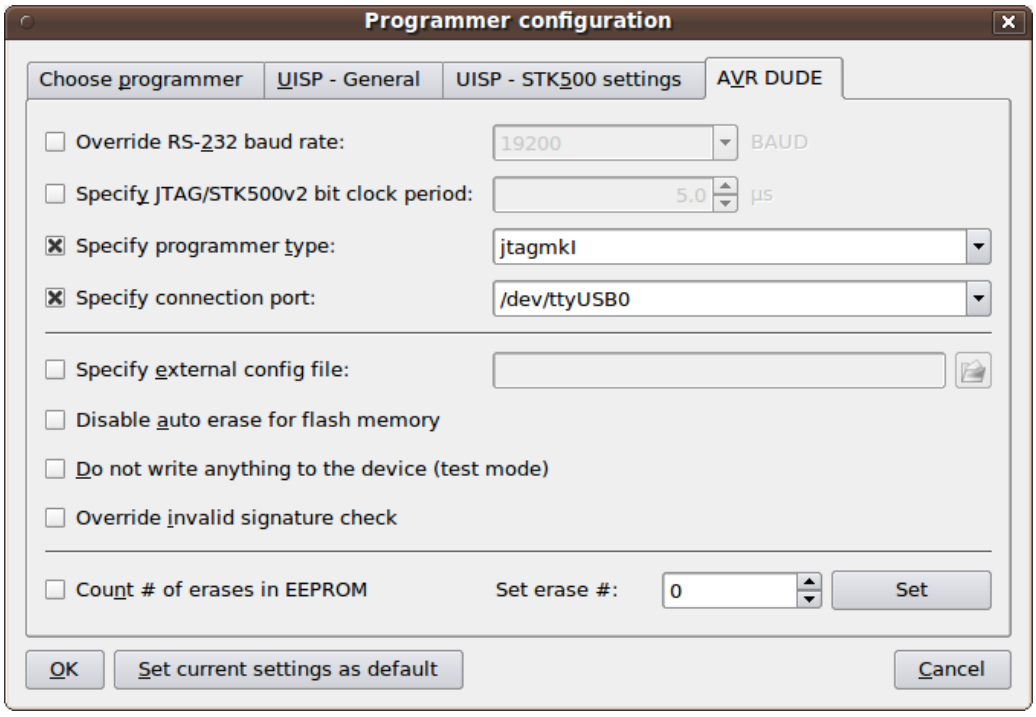
Achtung! Da die HomeLab Library nicht im *Linker* Konfigurationstab eingefügt werden kann, muss der Befehl “-lhomelab“an den Namen der MAP-Datei angehängt werden.



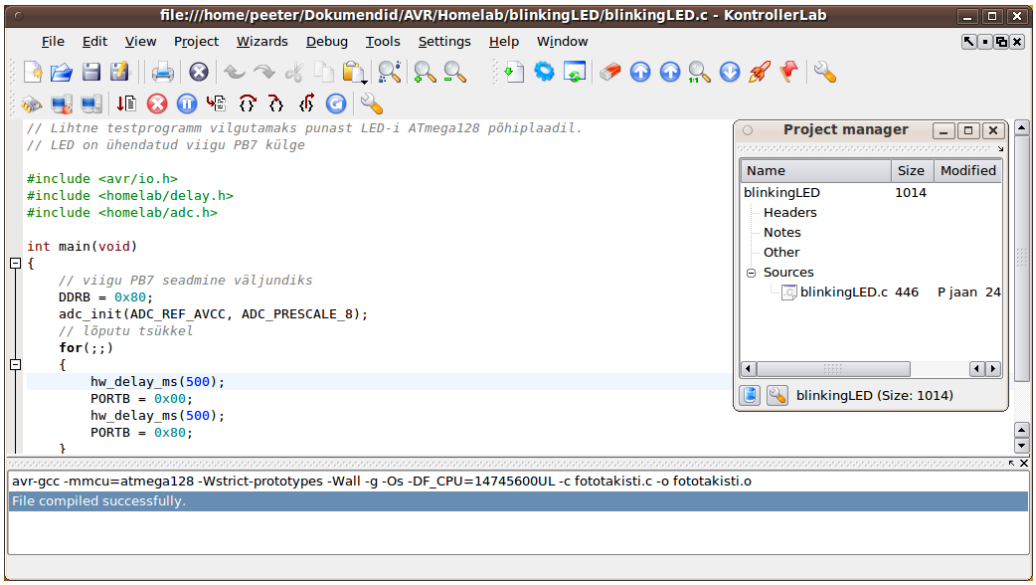
Wenden Sie die Einstellungen im *Compiler* Tab an, wie auf dem folgenden Screenshot gezeigt.Bevor Sie auf *OK* klicken, sollte die Compilerkonfiguration ebenfalls als Standard gesetzt werden.



4. Konfigurieren Sie das Programmiergerät durch öffnen von *Project* → *Configure Programmer*. Wählen Sie „AVRDUDE“als Programmiergerät. Setzen Sie im „AVRDUDE“Tab den Typ des Programmiergerätes auf *jtagmkI* und den Anschlussport auf */dev/ttyUSBx* (wobei „x“der Portindex ist.). Stellen Sie diese Konfiguration ebenfalls als Standard ein.



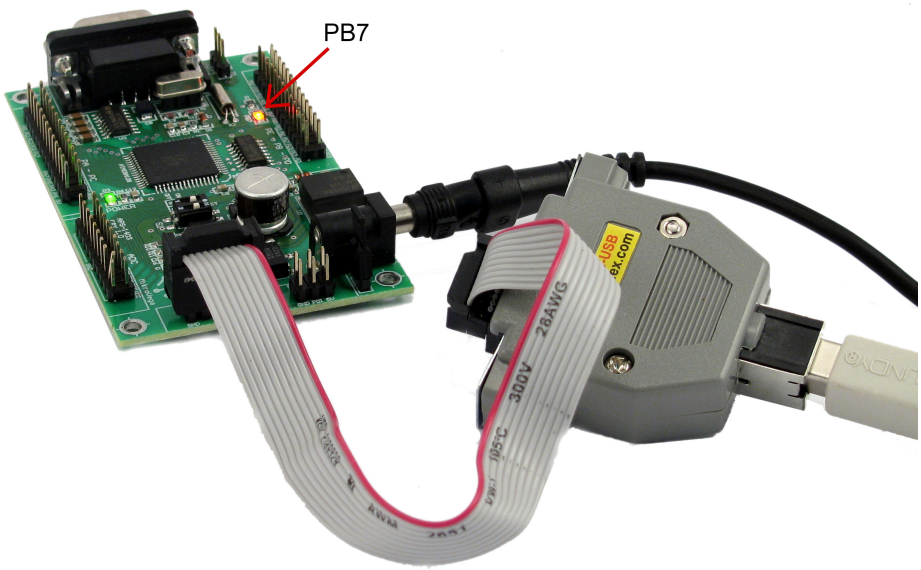
5. Ordnen Sie schließlich die Fenster im KontrollerLab so an, wie es angenehm ist und bereiten Sie sich darauf vor, das erste Programm zu schreiben.



Ein Programm schreiben und testen

Nachdem die Konfiguration abgeschlossen ist, ist es Zeit zu programmieren.

1. Schließen Sie das Programmiergerät am Controllermodul an. Überprüfen Sie noch einmal ob das Programmiergerät richtig am JTAG Anschluss angeschlossen ist. Das Kabel muss von der Platine weg zeigen (siehe Abbildung). Schließen Sie nun die Stromversorgung an und überprüfen ob die grüne LED leuchtet.



2. Schreiben Sie das folgende Programm in das Dateieditor-Fenster des KontrollerLab und kompilieren Sie es:

```
#include <avr/io.h>
#include <homelab/delay.h>

int main(void)
```

```
{
  // Setze Pinn PB7 als output
  DDRB = 0x80;

  // Loeputu tsuekkel
  while (true)
  {
    // Invertiere PB7
    PORTB ^= 0x80;
    hw_delay_ms(500);
  }
}
```

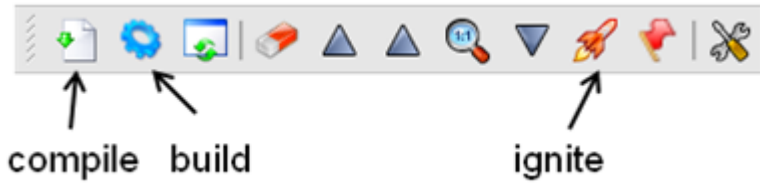


Abbildung 6.3: Toolbar

Stellen Sie sicher, dass das Outputfenster die Nachricht „File compiled successfully“anzeigt.Falls eine “Error(s) Occurred:“Nachricht erscheint, überprüfen Sie den Programmcode auf Fehler.

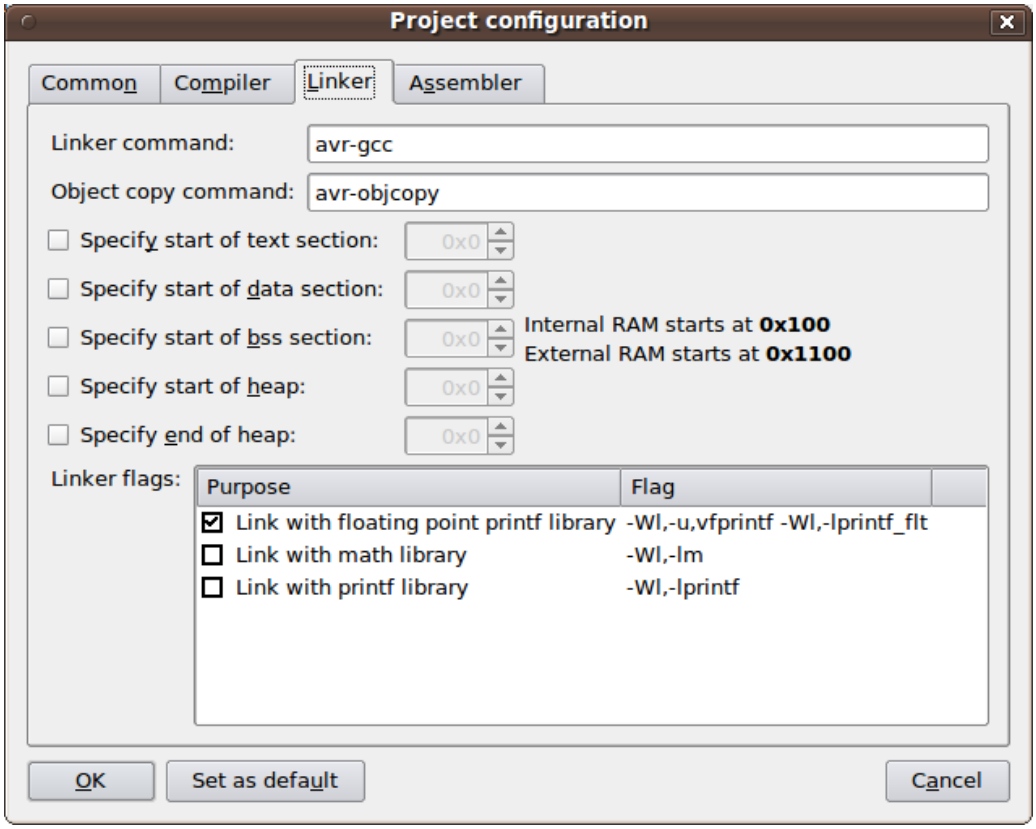
3. Um das Programm in den Controller zu laden wählen Sie *Ignite*. Wurde es erfolgreich in den Controller geladen, sollte die Nachricht „Project built and uploaded successfully“erscheinen.

Falls alles richtig gemacht wurde, sollte die rote LED auf dem Controller in 1-Sekunden Intervallen blinken.

Nutzung von floating-point Variablen

Manchmal ist in es in AVR Programmen notwendig, floating-point Variablen zu nutzen. Um mit ihnen zu rechnen und sie mit Funktionen vom Typ *printf* zu nutzen, müssen die folgenden Anpassungen in der Projektkonfiguration vorgenommen werden:

1. Öffnen Sie *Project Configurations* sowie den Tab *Linker*. Überprüfen Sie die erste Zeile der Liste *Linker flags* (siehe Abbildung).



2. Klicken Sie auf *OK* und schließen Sie das Fenster.

6.2 Digitale Input/Output Pins

Die folgenden Kapitel stellen den digtalen Input/Output Pin dar, welche Grundfunktionen des Mickocontrollers sind. Input/Output Pins sind Mikrocontrollerkontakte. Es sind die sogenannten „Beine“(„legs“). Sie dienen dazu, digitale Signale zu empfangen und zu senden. Ist ein Pin als Input konfiguriert, kann der Status von Schaltern oder einfachen Sensoren, die mit ihm verbunden sind, von diesem Pin empfangen werden. Ist ein Pin als Output konfiguriert ist, kann er benutzt werden um LEDs oder elektrisches Anlagen zu steuern.

Fast alle typischen Mikrocontroller Pins erlauben die Ausführung einfacher Input-Output Funktionen, auch wenn diese Pins häufig noch alternative Funktionen haben. Zunächst sollen jedoch erst die Pins mit ihren Grundfunktionen dargestellt werden, bevor in separaten Kapiteln auf die alternativen Funktionen eingegangen wird.

6.2.1 Light-emitting Diode (LED)

Notwendiges Wissen: *Controllermodul,* *Digitales Input-Output Modul,* *Berechnung des LED-Widerstands,* *Register,* *Digitale Inputs/Outputs,* *Pins*

Theorie

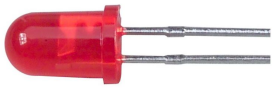


Abbildung 6.4: 5 mm legged LED

Leuchtdioden sind Halbleiter die Licht abstrahlen, wenn Strom in Durchlassrichtung durch die Diode fließt. Die Abkürzung LED steht für „light-emitting diode“. Es gibt verschiedenfarbige Leuchtdioden, mittlerweile ist sogar weißes Licht möglich. Wie jede normale Diode auch hat die LED 2 Kontakte - Anode und Kathode. In Zeichnungen ist die Anode mit einem “+“markiert und die Kathode mit einem “-“.

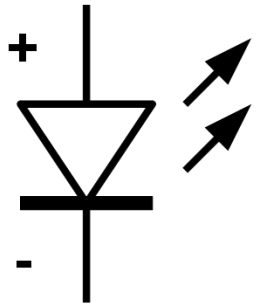


Abbildung 6.5: Schematische Darstellung einer LED and ihrer Polarität

Wenn Strom in Durchlassrichtung durch die LED fließt, ist die Anode mit der positiven und die Kathode mit der negativen Spannung verbunden. Die Spannung

der LED hängt von der Farbe der LED ab: lange Wellenlängen (rot) ˜2V, kürzere Wellenlängen (blau) ˜3V. Die Stromverbrauch einer LED ist in der Regel nicht mehr als ein paar Dutzend mW. Die Stromstärke darf daher nicht zu hoch sein, wenn größere Stromstärken oder Spannungen verwendet werden kann die LED zerstört werden.

Wenn die LEDs nur zur Beleuchtung verwendet werden, sollte man spezielle elektrische Schaltkreise verwenden, welche die Spannung und die Stromstärke für die LEDs regulieren.Jedoch werden LEDs sehr oft als Indikatoren genutzt und direkt vom Mikrocontroller mit Strom versorgt. Da die Betriebsspannung eines Mikrocontrollers höher ist, als die der LEDs, muss ein Widerstand in Serie mit der LED geschaltet werden um die Stromstärke zu regulieren und für den nötigen Spannungsabfall zu sorgen. Das Kapitel „Elektronik“enthält Anleitungen zur Berechnung des richtigen Widerstands.

LEDs werden in vielen Formen gefertigt. Die meisten LEDs haben runde eine Hülle mit 3mm oder 5mm Durchmesser und zwei lange metallische Pins. Der längere Pin ist die Anode, der kürzere die Kathode. „Surface mounted casing“LEDs (SMD - Surface Mounted Device) sind unten mit einem T gekennzeichnet, um so die Polarität anzugeben. Das Dach des T's steht für die Anode und der Fuß markiert die Kathode.

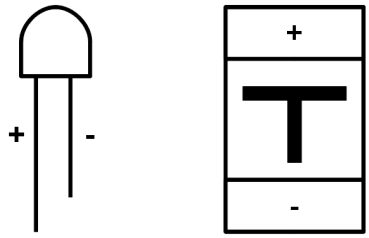


Abbildung 6.6: Polarität von legged und SMD LED's

HomeLab Übung 1

Das HomeLab Controller Controllermodul hat eine rote LED, dessen Anode über einen Widerstand an die +5V Stromversorgung angeschlossen ist und dessen Kathode am ATmega128 Pin PB7.Um die LED an- und auszuschalten muss PB7

als Output Pin definiert werden und entweder high oder low gesetzt werden. Wenn der Pin high gesetzt wird ist die LED aus, wird er low gesetzt ist die LED an. Grundsätzlich ist es möglich die LED so anzubringen, dass die Anode am PB7 angeschlossen wird und die Kathode geerdet wird (Widerstand nicht vergessen) -dann leuchtet die LED wenn der Pin high gesetzt ist und der LED ist aus wenn der Pin low ist.

Alle praktischen Beispiele für das HomeLab kit, inklusive LEDs schalten, nutzen die HomeLab Pin Bibliothek. Die Pin Bibliothek hat den Datentyp *pin*, welcher die Addressen des zum Pin gehörigen Register und der Bitmaske des Pins enthält. Wenn eine Pin Variable mit der Makrofunktion Pin im Programm erschaffen und initialisiert wird, kann der Pin mit dieser Variable im gesamten Programm genutzt werden, ohne Register benutzen zu müssen.Hier sind 2 Beispielprogramme, welche exakt das gleiche machen. Das eine Programm nutzt die HomeLab Bibliothek, das andere nicht.

```
//
// HomeLab Controllermodule LED Testprogramm.
// Der Cose basiert auf der HomeLab Bibliothek.
//
#include <homelab/pin.h>

//
// LED Pin Konfiguration.
//
pin debug_led = PIN(B, 7);

//
// Hauptprogramm
//
int main(void)
{
  // konfiguriert LED Pin als Output
  pin_setup_output(debug_led);

  // Aufleuchten der LED
  pin_clear(debug_led);
}
```

```
//
// HomeLab Controllermodul LED Testprogramm.
// Der Code greift direkt auf Register zu.
//
```

```
#include <avr/io.h>

//
// Hauptprogramm
//
int main(void)
{
  // Konfiguriert LED Pin als Output
  DDRB |= (1 << 7);

  // Aufleuchten der LED
  PORTB &= ~(1 << 7);
}
```

Das erste Beispiel nutzt die Pin Bibliothek (*pin.h* Datei). Zuerst wird eine Pin-Typ Variable *debug led* erschaffen, welche Informationen über den LED Pin enthält.Im Hauptprogramm wird dieser Pin mit *pin_setup_output* als Output gesetzt.Danach wird der Pin low gesetzt: *pin_clear*. Das Ergebnis: die LED leuchtet.Im zweiten Beispiel werden keine Variablen genutzt. Um die LED als Output zu markieren und zum Leuchten zu bringen, werden die Datenrichtung des Port B und die Output Registerwerte verändert. Der Leser, der mehr über den AVR weiß bemerkt, dass in beiden Beispielen kein Befehl nötig ist um die LED zum leuchten zu bringen, da der Standardoutputwert des AVR 0 ist. Aus Gründen der Vollständigkeit wird es jedoch hier beschrieben.

Was ist der Unterschied zwischen dem Gebrauch von Bibliothek und Register? Der Unterschied liegt im Komfort - Die Bibliothek ist einfacher, weil man die Registernamen und deren Auswirkungen nicht kennen muss. Der entscheidende Vorteil der Bibliothek ist jedoch die Anpassungsfähigkeit.Arbeitet man mit Registern müssen im Programm stets die Registernamen und Bitmasken geändert werden um einen Pin zu ändern. Nutzt man die Bibliothek muss dieses nur am Anfang, bei Initialisierung der Variable erfolgen. Die Nutzung von Registern hat jedoch einen entscheidenden Vorteil: Die Nutzung eines Pins erfolgt direkt und nicht über den Programmspeicher und zeitfressenden Funktionen. Jedoch sind die neuren AVR-GCC Kompiler Versionen so schlau, dass sie die Bibliotheksfunktionen zu direkten Befehle für die Registermanipulation transformieren, so als würde man die Register direkt im Programm ansprechen. Die Compiler können den Code aber nur dann optimieren wenn sie mit konstanten Variablen arbeiten und nicht mit unbeständigen Variablen die sich während des Programms

oder mit Arrays ändern.

Der folgende Programmcode ist ein Teil der Pin Operations Bibliothek, um die Vorgehensweise mit Pin Variblen zu erklären. Es wird für Anfänger nicht einfach sein zu verstehen, da C-Sprache Pointer genutzt werden, die nicht in diesem Buch beschrieben sind. Es gibt jedoch sehr viel Material über Pointer in Büchern und natürlich im Internet.

```
// Makrokonstante um Register-Pointer zu definieren
#define _REG_PTR_ volatile uint8_t *

//
// Pin Datentyp
//
typedef struct pin
{
  _REG_PTR_ ddr;
  _REG_PTR_ port;
  _REG_PTR_ pin;
  uint8_t mask;
}
pin;

//
// Makrofunktion für Initialisierung der Pin Variable
//
#define PIN(port_char, bit_index) \
{ \
  (_REG_PTR_)\ ## port_char, \
  (_REG_PTR_)\ ## port_char, \
  (_REG_PTR_)\ ## port_char, \
  bit_mask(bit_index) \
}

//
// Konfiguriere Pin als Output
//
inline void pin_setup_output(pin pin)
{
  bitmask_set(*pin.ddr, pin.mask);
}

//
// Setze Pin high
//
```

```
inline void pin_set(pin pin)
{
  bitmask_set(*pin.port, pin.mask);
}

//
// Setze Pin low
//
inline void pin_clear(pin pin)
{
  bitmask_clear(*pin.port, pin.mask);
}
```

HomeLab Übung 2

Zusätzlich zum Controllermodul befinden sich LEDs auch auf der digtalen I/O Modulplatine.Sie sind elektrisch genau so angeschlossen wie auf dem Controllerboard, die Kathode ist also am AVR Pin angeschlossen. Die rote LED ist am PC5 Pin angeschlossen, die gelbe am PC4, und die grüne am PC3. Das auf der HomeLab Bibliothek basierende Beispielprogramm sieht folgendermaßen:

```
//
// HomeLab Digitales i/o Modul LED Testprogramm.
//
#include <homelab/pin.h>

//
// LED Pin Konfiguration.
//
pin led_red    = PIN(C, 5);
pin led_yellow = PIN(C, 4);
pin led_green  = PIN(C, 3);

//
// Hauptprogramm
//
int main(void)
{
  // Konfiguriere LED Pins als Output
  pin_setup_output(led_red);
  pin_setup_output(led_yellow);
  pin_setup_output(led_green);
```

```
  // Aufleuchten der grünen LED
  pin_clear(led_green);

  // Ausschalten der roten und gelben LED
  pin_set(led_red);
  pin_set(led_yellow);
}
```

~Diskussion~

6.2.2 Schalter

Notwendiges Wissen: *User Interface Module,* *Registers,* *Digital Inputs/Outputs,* *Pins,* *Light-emitting Diode*

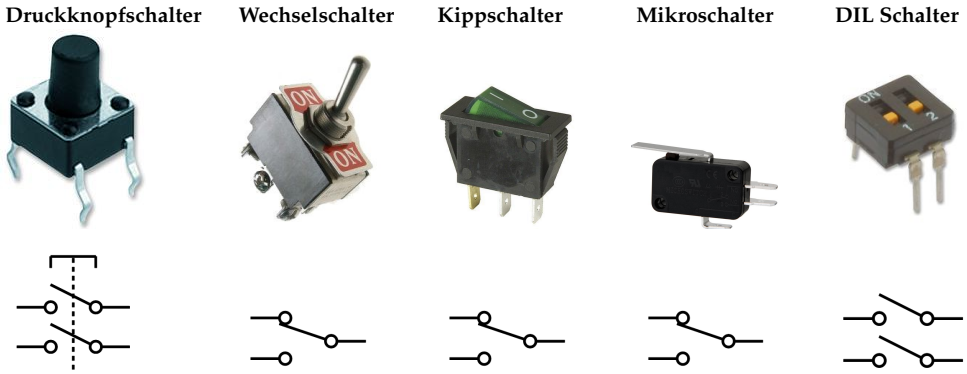
Theorie

Ein Schalter ist ein elektromagnetisches Gerät, welches einen elektrischen Schaltkreis verbindet oder trennt. Es gibt viele Typen von Schaltern, die am weitesten verbreiteten sind einfache mechanische Schalter, bei denen elektrische Anschlüsse mechanisch geschaltet werden.Sind die Anschlüsse verbunden, wird der elektrische Schaltkreis geschlossen und Strom kann durch den Schalter fließen. Wenn die Anschlüsse nicht verbunden sind, kann keine Elektrizität fließen.

Schalter werden normalerweise benutzt um elektrische Schaltkreise zu schalten, sie können jedoch auch als Sensoren benutzt werden. Der Schalter als Sensor ist auch Teil dieser Übung, Hochspannungsschalter werden jedoch ausgelassen. Schalter können anhand der Anzahl von Kontakten und ihrer Verbindungsmethode unterschieden werden. Unterschiedliche Typen sind: zwei-Kontakt Schalter und Doppelschalter, bei denen Kontaktpaare verbunden werden, darüber hinaus auch Druckknopf-, Schiebe-, Kipp- und Wechselschalter, ebenso wie Schalter die Verbindungen trennen anstatt zu verbinden.

Es werden unterschiedliche Schaltbildsymbole genutzt um die verschiedenen

Typen von Schaltern zu identifizieren. Unterhalb sind Beispiele typischer elektrischer Schalter in einem Schaltbild:



Um einen Schalter als Sensor mit einem Mikrocontroller zu nutzen, muss ein Kontakt des Schalters mit dem Pin des Mikrocontrollers verbunden sein und dieser muss als Input im Programm definiert werden. Wenn der Kontakt mit der Masse oder einem Inputpotential verbunden ist, wir die Bus Bitrate des Mikrocontroller-Pins geändert. Es erscheint logisch einen Wechselschalter zu nutzen, welcher es erlaubt einen Kontakt an den gewünschten Kontakt anzubringen (hier Masse oder Input). Für unseren Zweck ist es nicht ganz so einfach, denn im Moment des Schaltens sind die Kontakte nicht verbunden. Der Moment an sich ist sehr kurz (ms), aber in diesem Moment ist der Input-Pin des Mikrocontrollers nicht verbunden und hat daher einen unendlichen Wert.Auf Grund elektromagnetischer Störung (welche überall existiert) kann der Input-Pin der nicht angeschlossen ist, zu jeder Zeit zufällig einen Wert von 0 oder 1 haben.

Die Störanfälligkeit macht die Nutzung von Schaltern kompliziert. Die am häufigsten verwendete Methode um dies zu verhindern, ist den Input des Microcontrollers durch einen Widerstand an die Erde oder das Inputpotential anzuschließen. Einen so benutzten Widerstand nennt man einen pull-down oder pull-up Widerstand. Normalerweise liegt der Widerstand eines pull-down oder pull-up zwischen 1kΩ bis 1 MΩ. Wenn der Schalter offen ist, wird der Input mit der Spannung des Widerstands geladen, wenn man den Schalter schließt, empfängt der Input die Spannung des Schalters, da der Leitungswiderstand des Schalters im Vergleich zum Widerstand beinahe 0 ist. Allgemein kann er als „volt-box“bezeichnet werden.

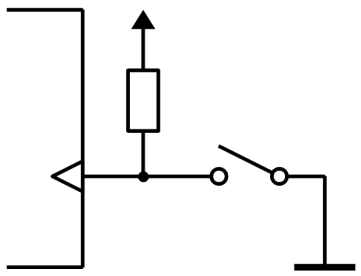


Abbildung 6.7: Schalter-Anschluss Schema mit einem pull-up Widerstand

Ein einfacher zwei-Kontakt Schalter kann als Sensor mit einen pull-up oder pull-down Widerstand genutzt werden. Hierzu wird ein Kontakt des Schalters mit dem Input verbunden und der andere mit dem Widerstand. Normalerweise haben Mikrocontroller eingebaute pull-up oder down Widerstände, daher ist es nicht notwendig einen Widerstand in den Schaltkreis einzubauen.AVR Mikrocontroller haben beispielsweise 20 kΩ - 50 kΩ pull-up Widerstände an ihren I/O Pins.Darüber hinaus muss noch erwähnt werden, dass mechanische Schalter ein weiteres Problem haben, den „switch bounce“. Hierdurch wird beim Schalten in sehr kleinen Abständen die Verbindung durch die Schwingung des mechanischen Schalters unterbrochen. Die Beispiele dieses Kapitels werden nicht durch den „switch bounce“beeinflusst, das Problem wird im nächsten Kapitel ausführlicher bearbeitet.

Übung

Es gibt drei Druckknopf-Schalter am digitalen I/O Modul. Diese verbinden die Pins den Mikrocontrollers mit der Erde, aber nicht direkt durch einen Widerstand. Auf diese Weise soll ein Kurzschluss vermieden werden, wenn der Knopf gedrückt wird, während der Pin als Output definiert wird. Die Schalter besitzen einen pull-up Widerstand, welcher aber einen viel größeren Widerstand als die Schutzwiderstände hat, daher liegt beim Drücken des Schalter eine Spannung von nahezu 0 V am betreffenden Pin an.

Die Schalter befinden sich an den PC0, PC1 und PC2 Pins. Um den Status der Schalter zu lesen müssen die betreffenden Pins des Mikrocontrollers als Input gesetzt werden. Man benötigt nicht die internen pull-up Widerstände des AVR, da die Pins schon externe Widerstände haben.Wenn der Schalter betätigt wird, hat der entsprechende Bus des Pins den Wert 0, wird er losgelassen, den Wert 1. Mit Hilfe von LED Indikatoren kann überprüft werden, ob der Mikrocontroller den Status den Schalters verstanden hat.

Der Beispielcode zur Benutzung der Schalter basiert auf der HomeLab Pins Bibliothek, welche im LED Kapitel eingeführt wurde.

```
//
// Program zum Testen der Schalter des digitalen I/O Moduls
//
#include <homelab/pin.h>

//
// Festlegung der Pins von LED-d sowie der Schalter.
//
pin leds[3]    = { PIN(C, 5), PIN(C, 4), PIN(C, 3) };
pin buttons[3] = { PIN(C, 2), PIN(C, 1), PIN(C, 0) };

//
// Hauptprogramm
//
int main(void)
{
  unsigned char i;

  // Setzt die LED Pins als Output und die Schalter-Pins als Input.
  for (i = 0; i < 3; i++)
```

```
  {
    pin_setup_output(leds[i]);
    pin_setup_input(buttons[i]);
  }

  // Endlosschleife
  while (true)
  {
    // Jeder Schalter hat eine zugehörige LED,
    // die aufleuchtet, wenn er betätigt wird.
    for (i = 0; i < 3; i++)
    {
      pin_set_to(leds[i], pin_get_value(buttons[i]));
    }
  }
}
```

Im Beispiel werden LEDS und Knöpfe als Array definiert - das ermöglicht es Sie in einer *for* Schleife zu nutzen. Wird das Programm gestartet, werden die LED Pins als Output und die Schalter als Input gesetzt. In der Endlosschleife des Programms wird durchgehend der Status der Schalter abgefragt, welcher den Status der dazugehörigen LED bestimmt. Der erste Schalter ist für die grüne LED, der Zweite für die Gelbe und der Dritte für die Rote.

6.2.3 Vermeidung von Kontaktprellung

Vajalikud teadmised: *User Interface Module,* *Digital Inputs/Outputs,* *Pins,* *Delay,* *Switch*

Theorie

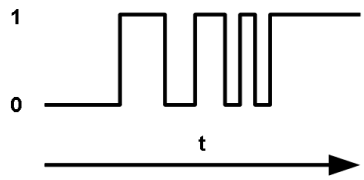


Abbildung 6.8: Kontaktprellung eines Schalters

Wie bereits im einleitenden Kapitel zu Schaltern erwähnt, gibt es im Ungang mit mechanischen Schaltern den Effekt des Prellens (ungewolltes, kurzzeitiges, wiederholtes Öffnen und Schließen eines Kontaktes bei dessen

Betätigung). Dieses Problem wird durch die Elastizität des Metalls, aus dem die Schalter bestehen, verursacht. Im Moment des Ein- oder Ausschaltens prellen die Kontakte, was zu einer Vielzahl falscher Schaltungen führt. Die Anzahl und Dauer der Schaltungen hängt von dem Schalter ab, liegt aber gewöhnlich im Bereich von wenigen Millisekunden. Für den Fall, dass ein Schalter genutzt wird, um ein elektronisches Gerät zu starten, hat dieses Problem keine großen Auswirkungen. Wird er jedoch zur Steuerung eines Gerätes verwendet kann mehrfaches Schalten das Gerät beschädigen.

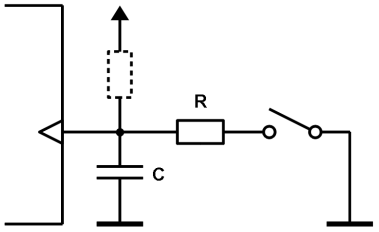


Abbildung 6.9: RC-filter eines Schalters

Die bekannteste Methode zur Vermeidung von Kontaktprellung ist die Filtrierung. Dieses kann elektrisch oder durch Software erfolgen. Um elektrisch zu filtern muss der Schalter an einen Tiefpassfilter angeschlossen werden - zum Beispiel an einen RC Filter - welcher die Spannungschwankungen beseitigt sodass der Pin des Mikrocontrollers keine schwankenden Werte erhält. Ein solcher RC Filter ist auf der Zeichnung abgebildet. Filtrierung mittels Software wird durchgeführt, indem dem Pin, an welchem der Schalter angeschlossen ist, ein Wert zugewiesen wird. Bleibt dieser Wert zu verschiedenen Zeitpunkten innerhalb eines zuvor gesetzten Zeitlimits gleich, kann angenommen werden, dass der Schalter stabil ist und kein Flimmer-Problem aufweist. Es muss jedoch bei jeder Art der Filtrierung ein Verzögerungsfaktor bei Definition des Status berücksichtigt werden.

Praktisches Beispiel

Elektrische Filtrierung wird für die HomeLab Schalter nicht verwendet, da hier die Beseitigung von Fehlschaltungen mit Hilfe von Software geübt werden soll. Die Übung besteht aus zwei Teilen. Das Ziel des ersten Teils ist, das Prellen der Schalter des digitalen Input/Output Moduls zu zeigen. Hierzu wird das folgende Programm genutzt; durch jede Betätigung des Schalters wird die nachfolgende LED aufleuchten. Eine Fehlschaltung lässt die LEDs mehrfach und zufällig aufleuchten. Electrical filtering is not used on HomeLab switches, since it would not allow practicing the elimination of miss switching's with software. The exercise is in two parts. The goal of the first part is to demonstrate the bouncing of Digital i/o module switches. The following program is used for this; each pressing on the button will light the next LED in line. Wrongly pressed button will causes LEDs to light several times and it appears as the LEDs light randomly.

```
//
// Das Programm um Kontaktprellung am digitalen Input/Output Modul zu
    demonstrieren.
//
#include <homelab/pin.h>

//
```

```
// Festlegung der LED-Pins und Schalter.
//
pin leds[3] = { PIN(C, 5), PIN(C, 4), PIN(C, 3) };
pin button  = PIN(C, 0);

//
// Hauptprogramm
//
int main(void)
{
  unsigned char new_value, old_value = 0;
  unsigned char index, counter = 0;

  // LED-Pins als Output setzen.
  for (index = 0; index < 3; index++)
  {
    pin_setup_output(leds[index]);
  }

  // Schalter-Pins als Input setzen.
  pin_setup_input(button);

  // Endlosschleife
  while (true)
  {
    // Status des Schalters auslesen.
    new_value = pin_get_value(button);

    // Kontolle, ob der Schalter heruntergedrückt wurde,
    // was bedeutet, dass der neue Status 1 ist, der alte 0.
    if ((new_value) && (!old_value))
    {
      // Erweiterung des Lesegerätes und Nutzungvon Modul 3
      counter = (counter + 1) % 3;

      // Aufleuchten der LED, welche mit dem Wert des Lesegerätes ü
   bereinstimmt.
      for (index = 0; index < 3; index++)
      {
        pin_set_to(leds[index], index != counter);
      }
    }

    // Berücksichtigung des alten Status.
    old_value = new_value;
```

```
  }
}
```

Es gibt diverse Softwarelösungen zur Filtrierung. Sie können einfach oder komplex sein, wobei jede ihre Vor- und Nachteile hat. Sieht das Programm nur wenige seltene Betätigungen des Schalters vor, kann eine lange Pause als Folge der Betätigung eingefügt werden. Auf diese Weise können durch Kontaktprellung verursachte Wirkungen auf die Schaltung vermieden werden. Jedoch muss bei Anwendung dieser Lösung bedacht werden, dass, sofern der Nutzer den Schalter für eine längere Zeit gedrückt hält, das Programm ebenfalls auf die Fehlschaltung, welche durch Lösen des Schalters hervorgerufen wird, reagiert. Ein Programm, welches den Status des Schalters in einer festgeleten Zeitperiode mehrfach überprüft ist zuverlässiger (das Ergebnis ist umso besser, je länger die Periode und je größer die Anzahl der Kontrollen ist). Nachfolgend ist eine Funktion abgebildet, um filtrierte Werte eines Schalters für das digitale Input/Output Mpdul auszulesen:

```
//
// Funktion zum Auslesen filtrierter Werte aus dem I/O
    Erweiterungsmodul.
//
unsigned char pin_get_debounced_value(pin button)
{
  unsigned char buffer = 0xAA;
  unsigned char timeout = 100;

  // Abwarten, bis der Status des Schalters We wait until the status of
     the button is celar or clearing the state expires
  while (timeout-- > 0)
  {
    // Having 8 place (bit) bufffer of state.
    // All previous states (bits) are shifted to left
    // and a new state(bit) is added to the right.
    buffer <<= 1;
    buffer |= (pin_get_value(button) ? 0x01 : 0x00);

    // If all 8 bits are high, then the button is definitely pressed
    down
    if (buffer = 0xFF)
    {
      return 1;
    }

    // If all 8 bits are low, then the button is definitely up.
    if (buffer = 0x00)
    {
      return 0;
    }

    // 1 ms break.
    // This function can be found from the library of the HomeLab.
```

```
    sw_delay_ms(1);
  }

  // If we can not examine the state, then we assume that the button
    was not pressed.
  return 0;
}
```

Diese Funktion generiert eine Verzögerung durch Nutzung einer Funktion, die in der entsprechenden Übung erläutert wurde. Zu dieser Zeit müssen wir nur wissen, dass die Verzögerungsfunktion eine Verzögerung von 1 ms am Ende jedes Zyklus generiert, um den Status des Schalters lesen zu können. Befindet sich der Schalter 8 Lesezyklen lang in der gleichen Position, If the button is in the same position during 8 readings, it returns to the counted position. In case the button is unstable the entire procedure may take up to 100 ms. This function is included in the library of pins, hence there is no need to add it to the program for passing the example. In order to try this first part of the exercise, it has to be altered a little - include library of generating the delay in the program and at the point where the value of the button was read, directly apply the function with filter. The result is as follows:

```
//
// The program for filtering the debounce of buttons of Digital i/o
    module.
//
#include <homelab/delay.h>
#include <homelab/pin.h>

//
// Determining pins of LEDs and buttons.
//
pin leds[3] = { PIN(C, 5), PIN(C, 4), PIN(C, 3) };
pin button  = PIN(C, 0);

//
// Main program
//
int main(void)
{
  unsigned char new_value, old_value = 0;
  unsigned char index, counter = 0;

  // Setting the pins of LEDs as outputs.
  for (index = 0; index < 3; index++)
  {
    pin_setup_output(leds[index]);
  }

  // Setting the pins of button as input.
  pin_setup_input(button);
```

```
  // Endless loop.
  while (true)
  {
    // Reading the state of the button.
    new_value = pin_get_debounced_value(button);

    // Control whether the button was pressed down, that means,
    // is the new state 1 and the old state 0.
    if ((!new_value) && (old_value))
    {
      // Widening the counter and taking module number 3.
      counter = (counter + 1) % 3;

      // Lighting the LED witch corresponds to the value of the counter
    .
      for (index = 0; index < 3; index++)
      {
        pin_set_to(leds[index], index != counter);
      }
    }

    // Remember the old state.
    old_value = new_value;
  }
}
```

Testen wir das Programm nun, leuchten die LEDs in genau der Sequenz auf, in welcher der Nutzer den Schalter betätigt.

Aufgaben: Digitale I/O

Ziel ist es, ein Programm zu schreiben, welches die unten beschriebenen Aufgaben ausführt.

Aufwärm-Übung

* Durch Drücken von S1 leuchtet eine LED auf, drückt man S2 leuchten zwei und bei S3 drei LEDs.

Für Anfänger

1. Es wird eine von Hand betriebene Ampel an einer Fußgängerkreuzung simuliert. Solange kein Schalter gedrückt wird, leuchtet eine grüne LED für die Autos. Nach Betätigung eines zufälligen Schalters beginnt die grüne LED für drei Sekunden zu blinken, daraufhin leuchtet die gelbe LED für 3 Sekunden und die rote für 10 Sekunden. Nach dieser Sequenz leuchtet erneut die grüne LED.

2. Es soll gezählt werden, wie oft ein Schalter betätigt wurde. Nur durch ein vollständiges Lösen des Schalters wird das Herunterdrücken vollendet. Das Ergebnis wird als Binärcode auf den LEDs angezeigt. Das maximale Ergebnis bei drei LEDs ist 7 (23-1). Die grüne markiert das erste, die gelbe das zweite und die rote das dritte Bit.

3. Durch Drücken von Schalter 1 leuchten LED 1 und LED 3. Drückt man S2 leuchtet die gelbe LED, Schalter 3 sorgt dafür, dass keine LED leuchtet. Diese Operation muss unter direkter Verwendung von Registerwerten (also ohne Nutzung der HomeLab Bibliothek) durchgeführt werden.

4. Es soll gezählt werden, wie oft ein Schalter gedrückt wurde. Das Ergebnis wird durch Blinken der LEDs dargestellt. Nach jeder Betätigung des Schalters wird die Anzahl der blinkenden LEDs um eins erhöht. Der Schalter kann zufällig ausgewählt werden. Damit die LED blinkt muss eine Unterfunktion genutzt werden, deren Parameter die blinkenden LEDs zählt.

5. Wird S1 betätigt, sendet die rote LED durch Blinken „SOS“im Morsecode. Drückt man S2, sendet die gelbe LED „CQD“und durch Drücken von S3 zeigt die grüne LED „OK“.

Für Fortgeschrittene

1. Zu jedem der drei Schalter gehört eine LED, welche aufleuchtet wenn der Schalter betätigt wird. Zum an- oder ausschalten sollen Register und nur eine zuordnende Operation genutzt werden (Hinweis: Nutzen Sie Bitverschiebungen).

2. Für jede LED gibt es einen Schalter. Der Controller bringt die LEDs in zufälliger Reihenfolge zum Blinken und der Nutzer muss diese Sequenz wiederholen. Die Blinksequenz verlängert sich, nach jeder Runde kommt eine LED hinzu. Bei einem falschen Eintrag blinken alle LEDs dreimal (Die Anzahl korrekter Einträge des Nutzers muss auf einem LCD-Display angezeigt werden).

3. Das Programm misst die Zeit für eine Reaktion. Eine zufällige LED leuchtet auf und der Nutzer muss den zugehörigen Schalter so schnell wie möglich betätigen. Die Zeit bis zum Aufleuchten der LED ist zufällig, aber nicht kleiner als 100 ms. Das Ergebnis wird auf einem LCD-Display in Millisekunden dargestellt. Der Wert der Schalter kann nicht mit einer Filterfunktion für das Flackern gelesen werden, da diese eine weitere Verzögerung verursacht.

Fragen

1. Was ist der Unterschied zwischen den Operationen “=“und “=“? Nennen Sie zwei Beispiele um Ihre Antwort zu verifizieren.

2. Was ist der Unterschied zwischen den Operationen “|“und “||“? Nennen Sie zwei Beispiele um Ihre Antwort zu verifizieren.

3. Schreiben Sie einen Ausdruck, welcher die Gleichung “x = x + 1“zwölfmal verwendet.

4. Wie wird eine Endlosschleife in C geschrieben? Nennen Sie zwei Beispiele.

5. Welcher Variablentyp wird in C verwendet um positive Werte zwischen 7 und 154 dazustellen?

6. Welches Register legt die Richtung eines Anschlusses fest? Nennen Sie ein Beispiel zur Konfiguration von Input und Output eines Anschlusses.

7. Welche Methoden gibt es um das Zurückspringen von Kontakten zu sowie der durch Flimmern hervorgerufenen falschen Verbindungen zu verhindern? Geben Sie Lösungsbeispiele unter der Nutzung von Hard- und Software.

8. Was versteht man unter einer Bitverschiebung? Nennen Sie ein praktisches Beispiel und erklären Sie dieses.

9. Warum werden pull-up Widerstände in Schalter verwendet? Wie ist deren Widerstand festgelegt?

10. Errechnen Sie den Widerstand für die regulierende LED Stromstärke, die Spannung beträgt 5 V, die Durchlassspannung der LED beträgt 2,7 V und die Stromstärke 30 mA.

6.3 Timer und Verzögerungen

Dieses Kapitel beschreibt die Möglichkeit, die Zeit der Mikrocontroller durch Timer und Verzögerungen zu kontrollieren. Dies ist hilfreich für alle Anwendungen, die eine Zeitmessung benötigen, Unterbrechungen machen oder für präzise Steuerprozesse. Alle Timer und Verzögerungen sind ähnlich im Aufbau, jedoch können sie voneinander unterschieden werden. Timer sind physikalische Module des Mikrocontrollers, welche unabhängig vom Prozessor des Mikrocontrollers funktionieren, Verzögerungen sind Abschnitte eines Softwareprogramms welche dazu entwickelt werden, die Arbeitszeit des Prozessors zu nutzen. Beide arbeiten ähnlich, indem sie den Arbeitstakt des Microcontrollers zählen.

Beide Methoden haben ihre Vor- und Nachteile, welche in den entsprechenden Übungen besprochen werden. Anfangs waren Timer separate Schaltkreise, die nicht Teil eines Mikroprozessors waren. Auch heute gibt es noch immer separate Timerchips, da in einigen Anwendungen keine Mikrocontroller verwendet werden, und einige Timer effizienter sind, als die eines Mikrocontrollers.

Timer haben sich von einfachen Zeitzählgeräten zu komplexen System entwickelt, welche Signale empfangen und generieren. Sie modulieren und demodulieren Signale und ermöglichen, Taktsignale zu multiplizieren und dividieren. Darüber hinaus gibt es spezielle „Time-To-Digital“Konverter (TDC), welche Zeit in Picosekunden messen können, aber wir werden uns hier nur auf einfachere Timer konzentrieren.

6.3.1 Softwareverzögerungen

Notwendiges Wissen: *Controller module,* *Architecture,* *Pins,* *Delay*

Theorie

Oft ist es wichtig in einem Programm eines Mikrocontrollers Verzögerungen einzubauen, einige Aktionen zu terminieren, oder zu warten bis sie abgeschlossen sind. Die einfachste Methode die Arbeit eines Mikrocontrollers zu pausieren ist, ihn mit einer alternativen Operation zu überladen - z.B. durch einen Befehl, große Nummern zu zählen. Aus der Taktfrequenz des Prozessors kann man errechnen, wie weit das Programm zählen sollte um eine Verzögerung hervorzurufen.Eine Zahl von Null bis zum Wert der Taktfrequenz des Prozessors in Hz, sollte theoretisch eine Verzögerung von einer Sekunde erzeugen. Aus verschiedenen Gründen ist dieses jedoch praktisch nicht ganz so einfach.

Wenn der Prozessor des Microcontrollers mit Nummern rechnet, deren Binärform so groß ist wie der innere Bus (AVR hat 8 Bits), dann braucht es einen Taktzyklus des Prozessors um eine arithmetische Operation, wie z.B. 1 zu einem Wert zu addieren, durchzuführen. Soll mit tausenden oder millionen gerechnet werden, muss die Nummer 16 oder 32 Bit groß sein und ein 8-Bit Prozessor benötigt zur Berechnung mehr als eine Taktfrequenz. Daher sollte man sich mit der Arbeitsweise und insbesondere den Befehlen des Prozessors auskennen, wenn man mit großen Werten arbeitet.

Wenn man in fortgeschrittenen Sprachen programmiert (z.B. C), werden die Programme nicht auf Basis des Befehls geschrieben, um eine Softwareverzögerung zu erzeugen. Vielmehr muss man auch den Compiler kennen, welcher das Programm zu Maschinencode konvertiert. Davon hängt ab, wie viele Befehle (und Phasen) es braucht um eine arithmetische Operation durchzuführen. Das wird durch mehrere Gründe komplexer, durch die Fähigkeit des Compilers das Programm in Maschinencode umzuwandeln - z.B. dadurch, dass der Maschinencode für Speicherverbrauch optimiert, oder einfacher auszuführen wird. Diesen Compilervorgang nennt man Optimierung. Durch verschiedene Optimierungsmodi verändern sich die Maschinencodes für die Soft-

wareverzögerung sowie die Dauer der Verzögerung.

Beispiel

Das folgende Beispiel generiert eine Softwareverzögerung im AVR Mikrocontroller. Ein Teil des in C geschriebenen Programms zählt eine Variable x in vier Zyklen von 0 bis 100.In jedem Zyklus wird ein leerer „no-action empty“Befehl ausgeführt. Dieser wird benötigt, der Compiler einen Zyklus ohne Inhalt als nutzlos betrachtet und ihn daher aus dem Programm werfen würde.

```
unsigned char x;

// Zyklus bis x ist 100
for (x = 0; x < 100; x++)
{
  // Mit "empty"-Befehl nop
  asm volatile ("nop");
}
```

Dies ist der gleiche Teil des Programms nach dem Kompilieren. Die beiden Hexadezimalzahlen links geben den Maschinencode an, und rechts befindet sich der Befehl mit den Operanden in Assembler.Der Maschinencode und die Assemblersprache sind konform; Assembler dient nur dazu, den Maschinencode für Menschen lesbar zu machen. Kompiliert wurde mit der Optimierungsmethode für die Länge des Programms (Parameter -Os).

```
80 e0     ldi  r24, 0x00   ; r24 lädt Zahl 0 in den index
00 00     nop               ; "Empty" Operation
8f 5f     subi r24, 0xFF   ; subtrahieren von 255 aus dem r24 Index,
   was bedeutet +1 addieren
84 36     cpi  r24, 0x64   ; r24 Index mit der Zahl 100 vergleichen
e1 f7     brne .-8         ; Falls der Vergleich nicht übereinstimmt,
   8-Bits zurücktransferieren
```

In der kompilierten Form erkennt man, was mit dem Zyklus der C-Sprache passiert und man kann berechnen wie viele Taktfrequenzen benötigt werden, um den Zyklus einer Periode zu beenden. Die Information über die Wirkung der Befehle und ihre Laufzeit findet man im AVR Datenblatt. Im vorliegenden Beispiel, braucht es 4 Taktzyklen um 4 Befehle in einer Periode auszuführen, da jeder Befehl einen Takt benötigt. Zusätzlich wird ein Takt vor dem Zyklus benötigt, um die Befehle zu laden und ein Takt, um den Zyklus zu beenden.Wenn man nun den Arbeitstakt von 14,7456 MHz annimmt, kann man die vom Programm verursachte Verzögerung berechnen.

$(1 + 100 \cdot 4 + 1) \,/\, 14745600 = 27{,}26\ \mu s$

Die Verzögerung in diesem Beispiel ist in Mikrosekunden angegeben und die Variable nutzt 8-Bit, daher ist der Maschinencode recht einfach. Um eine Pause in Millisekunden zu verursachen, müssen weitaus größere Nummern gezählt werden, wodurch dann der Maschinencode auch länger wird. Es können ebenfalls Zyklen verwendet werden, die ineinander greifen, jedoch steht dann die Verzögerung nicht in linearer Relation mit der Nummer des Zyklus, da mit jedem Level des Zyklus eine kleine zusätzliche Verzögerung verursacht wird.

Das Ziel dieser Übung ist es nicht präzise Softwareverzögerungen in Maschinencode zu erstellen, da diese eine sehr präzise Arbeit ist und die Funktionen zur Erzeugung von Verzögerungen bereits in der avr-libc und der Bibliothek von HomeLab hinterlegt sind. Diese werden auch in den folgenden Beispielen genutzt.

Bei der Arbeit mit Softwareverzögerungen arbeitet, ist es wichtig zu wissen, dass diese trotz ihrer Einfachheit, eine sehr ineffiziente Methode sind was den den Stromverbrauch betrifft.Während jedes Taktes in dem der Mikrocontroller unnötig Zahlen zählt, wird Energie verbraucht. Wenn also batteriebetriebene Anwendungen genutzt werden, ist es nicht sinnvoll lange Softwareverzögerungen zu schreiben. Für diesen Fall sollte man lieber Hardwaretimer nutzen, welche unabhängig arbeiten und den Prozessor aus dem Ruhezustand holen wenn es an der Zeit ist die Arbeit fortzuführen.

Übung

Der folgende Programmcode beinhaltet die Softwareverzögerunsfunktion *sw_delay_ms*, welche einen gegebene Verzögerung in ms mit dem Parameter *count* verursacht. Die Funktionen nutzt die Funktion der Bibliothek avr-libc *_delay_ms* welche zum Teil in Assembler geschrieben ist.*_delay_ms* wird in dieser Übung nicht genutzt, da hiermit bei langen Verzögerungen Probleme auftreten können. *sw_delay_ms* erlaubt es aber bis zu 65535ms lange Verzögerungen ohne Komplikationen zu nutzen.

```
//
// Softwareverzögerung in Millisekunden.
//
void sw_delay_ms(unsigned short count)
{
  // Zählen der Variable der Verzögerung auf 0
  while (count-- > 0)
  {
    // 1ms Verzögerung mit spezieller Funktion.
    _delay_ms(1);
  }
}
```

Das folgende Programm dient dazu, die gegebene Funktion zu nutzen, es erzeugt zwei Verzögerungen in der Endlosschleife: 100 ms und 900 ms. Während der kürzeren Verzögerung wird eine LED eingeschaltet und während der längern ausgeschaltet. Das Resultat: die LED blinkt periodisch.

```
//
// Das Vorführprogramm des HomeLab für Softwareverzögerungen.
// Es lässt eine LED für ~1 Sekunde aufleuchten.
//
#include <homelab/pin.h>
#include <homelab/delay.h>

//
// Festlegung des Pins zum Testen der LED
//
pin debug_led = PIN(B, 7);
```

```
//
// Hauptprogramm
//
int main(void)
{
  // Setzt LED Pin als Output.
  pin_setup_output(debug_led);

  // Endlosschleife
  while (true)
  {
    // Aufleuchten der LED
    pin_clear(debug_led);

    // Softwareverzögerung für 100 ms
    sw_delay_ms(100);

    // Ausschalten der LED
    pin_set(debug_led);

    // Softwareverzögerung für 900 ms.
    sw_delay_ms(900);
  }
}
```

Auch wenn es so aussieht, als blinke die LED jede Sekunde, dauert es tatsächlich etwas länger, da das ansprechen der LED und die Verzögerungsfunktionen ein paar Taktfrequenzen des Mikrocontrollers benötigen.

6.3.2 Hardwareverzögerung

Notwendiges Wissen: *Controller module,* *Counters/Timers,* *Pins,* *Delay,* *Timers,* *Software delay*

Theorie

Neben der Softwareverzögerung gibt es auch Timer, um Unterbrechungen zu erzeugen. Timer sind Hardwarekomponenten, die mit einer bestimmten Frequenz hoch oder runter zählen. Die Taktfrequenz des Timers kann über die Frequenz des Mikrocontrollers oder einen externen Taktgeber erzeugt werden. Allgemein kann die Taktfrequenz mit einem Multiplikator dividiert werden um eine geringere Frequenz zu erreichen. Hierzu wird ein Vorteiler verwendet. Wichtig ist jedoch, dass die festgelegte Taktfrequenz des Timers linear zu der

Zeit ist. Die Zeit kann durch Multiplikation der Periode der Taktfrequent des Timers mit dem Wert des Timers berechnet werden.

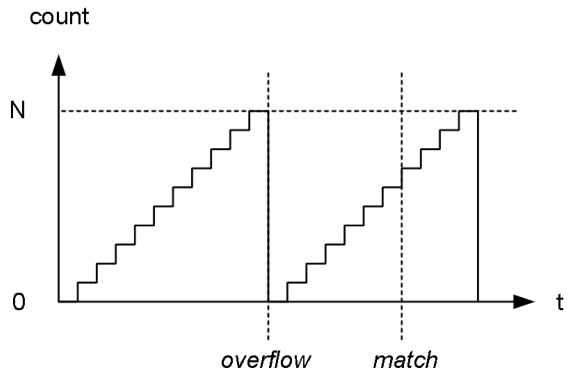


Abbildung 6.10: Durch Veränderung des AVR Timers hervorgerufene Ereignisse.

AVR Taktgeber können so eingestellt werden, dass sie bei Überlauf des Timers oder bei Vergleichstreffern informieren. Ein Überlauf tritt auf, wenn der Timer seinen maximalen Wert erreicht hat und der Zyklus wieder bei 0 anfängt. Nach Erreichen eines vorgegebenen Wertes beginnt der Timer bei jedem Anstieg seines Wertes diesen mit einem vom Benutzer vorgegebenen Wert zu vergleichen. Daraufhin werden die Bits des AVR-Statusindexes automatisch high gesetzt.

Um eine Verzögerung mit einem Timer zu generieren, muss man nur den Timer setzen und darauf warten, dass das Statusbit high wird. Im Unterschied zur Softwareverzögerung ist die Arbeit des Timers nicht vom Compiler abhängig, wodurch er zuverlässiger ist. Gleichzeitig ist die Diversität (bzw. Komplexität) der Einstellungen des AVR Timers hoch. Je nach Taktsignal des Mikrocontrollers kann es sein, dass es nicht exakt mit der gewünschten Verzögerungsperiode dividiert und die Verzögerung so ungenau wird.

Übung

Der Programmcode unterhalb stellt eine vereinfachte Timer-basierte Verzögerungsfunktion dar. Das Zählen funktioniert genauso wie bei einer Softwareverzögerungsfunktion. Es wird eine gewünschte Verzögerung von 1 ms mit dem Timer 0 eines 8-Bit ATmega128 erzeugt. Die zuvor errechnete Taktfrequenz liegt bei 14,7456 MHz und das Timersignal muss mindestens 64 Mal dividiert werden, so dass der Timer seinen Überlauf nicht in einer Millisekunde erreicht.Der Wert den der Timer haben muss, so dass der Überlauf nach 1 ms stattfindet wird mit durch die Variable *timer_start* dargestellt. *F_CPU*, eine Konstante in der Makrosprache, gibt die Taktfrequenz in Hz an. Diese sollte 25,6 betragen, da nur ganze Zahlen berücksichtigt werden, liegt der Startwert bei 26.Leider entsteht hier eine gewisse Ungenauigkeit, welche aber sehr gering ist (- 1,7μs).

In dem Zyklus wird der Timer initialisiert und die Überlauf-Flagge genullt (in dem eine 1 hineingeschrieben wird). Dann wird abgewartet bis der Timer vom Startwert bis 256 gezählt hat, also zum Überlauf. In dem Moment wird die Flagge hoch gesetzt und eine Verzögerung von einer 1 ms hat stattgefunden. Am Ende der Funktion wird der Timer gestoppt.

```
//
```

```
// Hardwareverzögerung in Millisekunden.
//
void hw_delay_ms(unsigned short count)
{
  // Berechnung des Startwertes des Timers.
  register unsigned char timer_start = 256 - F_CPU / 1000 / 64;

  // Start des Timers.
  timer0_init_normal(TIMER0_PRESCALE_64);

  // Zählen der Verzögerungsvariable auf 0.
  while (count-- > 0)
  {
    // Initialisierung des Timers.
    timer0_set_value(timer_start);

    // Überlauf-Flagge auf Null setzen.
    timer0_overflow_flag_clear();

    // Warten auf den Überlauf.
    while (!timer0_overflow_flag_is_set())
    {
      asm volatile ("nop");
    }
  }

  // Überlauf-Flagge auf Null setzen.
  timer0_overflow_flag_clear();

  // Timer anhalten.
  timer0_stop();
}
```

Das folgende Programm ist ähnlich wie das für die Softwareverzögerung. In der kürzeren 100 ms langen halb-Periode wird die LED eingeschaltet und in der längeren 900 ms halb-Periode ausgeschaltet. Das Resultat: die LED blinkt jede Sekunde. Leider beträgt die Periode in diesem Beispiel ebenfalls nicht exakt eine Sekunde, da die Ausführung anderer Funktionen des Programms ebenfalls Zeit verbraucht. Für ein exaktes Timing, muss ein 16-Bit Timer mit Interrupts genutzt werden.

```
//
// Beispielprogramm für eine Hardwareverzögerung mit dem HomeLab.
// Das Programm lässt eine LED immer nach ~1 Sekunde blinken.
//
#include <homelab/pin.h>
#include <homelab/delay.h>
```

```
//
// Festlegung des Pins der Test-LED.
//
pin debug_led = PIN(B, 7);

//
// Hauptprogramm.
//
int main(void)
{
  // LED Pin als Output setzen.
  pin_setup_output(debug_led);

  // Endlosschleife.
  while (true)
  {
    // Aufleuchten der LED.
    pin_clear(debug_led);

    // Hardwareverzögerung von 100 ms.
    hw_delay_ms(100);

    // Ausschalten der LED.
    pin_set(debug_led);

    // Hardwareverzögerung von 900 ms.
    hw_delay_ms(900);
  }
}
```

6.3.3 Periodische Interrupts

Notwendiges Wissen: *Controller module,* *User Interface Module,* *Interrupts,* *Counters/Timers,* *Pins,* *Delay,* *Timers,* *Software delay*

Theorie

Ziel dieses praktischen Beispiels ist es, die Verwendung von Interrupts an Beispiel von Timern darzustellen. Interrupts sind Programmteile welche auf Ereignisse, die in einem Mikrocontroller stattfinden, reagieren. Normalerweise dienen sie dazu, schnell auf ein Event zu reagieren. Sie können jedoch auch dazu genutzt

werden, um mehrere parallel ablaufende Vorgänge oder zeitlich abgestimmte Aktionen auszuführen, oder zur Energieeinsparung.Es ist beispielsweise möglich eine LED durch Interrupts blinken zu lassen, so dass die Blinkfrequenz nicht von einem Programm abhängig ist.

Übung

Das folgende Programm zeigt wie Timer eingestellt werden, um einen Interrupt auszulösen.Das Programm nutzt zwei LEDs des digitalen I/O Moduls. Der Status der roten LED wird mit einer Softwareverzögerung periodisch geändert, der Status der grünen LED wird geändert wenn ein Interrupt auftritt. Es gibt ein separates Beispiel für blinkende LEDs durch Softwareverzögerung, daher wird es an dieser Stelle nicht näher erläutert. Ziel ist es, die Bibliothek für Counter und Interrupts zu verwenden.

Zu Beginn des Programms wird der 16-Bit Timer 1 mit der Funktion *timer1_init_ctc* eingestellt. Dadurch wird der Timer in den CTC *clear timer on compare match* Modus gesetzt, wodurch sein maximaler Wert nicht 216 - 1 ist, sondern vom Benutzer eingestellt werden kann. In diesem Fall wird der maximale Wert so gesetzt dass er gleich dem Wert des ICR1 Indexes ist.Der Teiler des Timers ist 1024 und der Wert des ICR1 beträgt 14400. Bei einer Taktfrequenz von 14,7456MHz umfasst eine Periode somit genau eine Sekunde. Dieses kann mit folgender Formel berechnet werden:

f = 14745600 Hz / 1024 / 14400 = 1

Nachdem zugelassen wurde, dass ein Interrupt den maximalen Wert von Counter 1 erreicht, muss ein Interrupt auch global, also im Mikrocontroller, zugelassen werden. Dazu gibt es die Funktion *sei*, um sie zu untersagen die Funktion *cli*. Um den Programmteil für diese Funktionen und Interrupts festzulegen muss die Headerdatei *avr/interrupt.h* eingefügt werden.Ein Interrupt wird im Programm durch die Makrofunktion ISR definiert, deren Parameter der Name des Interruptvektors ist. Im folgenden Beispiel beträgt der Interruptvektor des maximal erreichbaren Wertes von Counter 1 *TIMER1_CAPT_vect*.

```
//
// HomeLab Beispiel für blinkende LED aufgrund von Counter-Interrupts.
// Zum Vergleich blinkender LEDs aufgrund von Interrupts,
// gibt es parallel eine durch Softwareverzögerung blinkende LED.
//
#include <homelab/pin.h>
#include <homelab/delay.h>
#include <homelab/timer.h>
#include <avr/interrupt.h>

//
// Festlegung der Pins der LEDs.
//
pin led_red   = PIN(C, 5);
pin led_green = PIN(C, 3);

//
// Interrupt
//
```

```
ISR(TIMER1_CAPT_vect)
{
  // Status der grünen LED ändern.
  pin_toggle(led_green);
}

//
// Hauptprogramm.
//
int main(void)
{
  // LED Pins als Output setzen.
  pin_setup_output(led_red);
  pin_setup_output(led_green);

  // Timer in den CTC Modus setzen.
  timer1_init_ctc(
    TIMER1_PRESCALE_1024,
    TIMER1_CTC_TOP_ICR);

  // Maximaler Wert des Timers ist 14400, wodurch
  // die Periode 1 s lang ist.
  // Formel: 14,7456Mhz / 1024 = 14400
  timer1_set_input_capture_value(14400);

  // Dem Interrupt erlauben, den Wert zu erreichen.
  timer1_input_capture_interrupt_enable(true);

  // Globalen Interrupt zulassen.
  sei();

  // Endlosschleife.
  while (true)
  {
    // Softwareverzögerung 1000 ms.
    sw_delay_ms(1000);

    // Status der roten LED ändern.
    pin_toggle(led_red);
  }
}
```

Zu Beginn des Programms sieht man, dass unabhängig davon was der Mikrocontroller gerade im Hauptprogramm macht, Interrupts stattfinden und die grüne LED blinkt.

Läuft das Programm ein paar Minuten wird ein entscheidender Aspekt deutlich, der im Rahmen der Softwareverzögerung nicht so einfach zu erkennen ist. Auch wenn die Verzögerung der blinkenden roten LED 1000 ms beträgt, ist die tatsächlich benötigte Zeit um einen Zyklus zu beenden länger. Das liegt daran, dass die Änderung vom LED-Status', die Angabe der Verzögerungsfunktion und das Beenden vom Zyklus einige Taktzyklen des Prozessors dauern. Daher sieht es so aus, als blinke die rote LED immer nach der grünen. Aus diesem Grund sollten Taktgeber oder andere präzise Vorgänge nicht mit Verzögerungen, sondern mit Timer-Interrupts ausgeführt werden.

6.3.4 Aufgaben

Ziel ist es, ein Programm zu schreiben welches die unten genannten Aufgaben durchführt.

Aufwärm-Übung

Lassen Sie die rote LED blinken. Die Periode soll 10 Sekunden (Halbperiode 5 Sekunden) betragen. Nutzen Sie eine Softwareverzögerungsfunktion, mit der Sekundenanzahl als Parameter.

Für Anfänger

1. Zeigen Sie mit einer Genauigkeit von 100 ms auf dem LCD Display die Zeit zwischen der Betätigung von 2 beliebigen Schaltern an. Die Art der Zeitmessung können Sie frei bestimmen.

2. Wird Schalter S1 betätigt, wird die Blinkgeschwindigkeit aller 3 LEDs zweimal reduziert. Durch Betätigung von S3, blinken die LEDs doppelt so schnell, und S2 setzt die Blinkfrequenz auf 1 Hz. Nutzen Sie Verzögerungsfunktionen oder Interrupts (die Interrupts sind schwieriger, aber bei der Verwendung von Verzögerungsfunktionen wird eine zusätzliche Verzögerung durch die Filterfunktion der Schalter auftreten.)

3. zeige die Druckfrequenz von Schalter S1 auf dem 7 Segment LED-Display in Hz-s. Die Anzeige der Frequenz muss nach unten mit 0 und nach oben mit 9 begrenzt werden.

4. Wird Schalter S1 gedrückt wird, zählt das Programm die Sekunden von 60 bis 0 herunter und schaltet dann die rote LED ein. Bei S2 beträgt die Zeit nur 30 s, anschließend wird die gelbe LED eingeschaltet. Die Zeit bei Betätigung von S3 beträgt 10 s, dann leuchtet die grüne LED. Alle Prozesse müssen gleichzeitig stattfinden. Die LEDs schalten sich aus, wenn der dazugehörige Schalter betätigt wird.

5. Zeigen Sie die Zeit auf dem LCD wie folgt an: hh:mm:ss. Nutzen Sie den Hardwaretimer 1 mit Interrupts, die Zeit muss mit den Schaltern eingestellt werden können. Es sollten drei Schalter genutzt werden, S1 erhöht die Stunden, S2 die Minuten, und S3 die Sekunden.

Für Fortgeschrittene

1. Entwickeln Sie eine Stoppuhr, die Stunden, Minuten, Sekunden, und Millisekunden auf dem LCD anzeigt. Schalter S1 startet die Zeit, S2 hält sie an, und S3 nullt die Uhr. Es müssen Interrupts des Timers genutzt werden.

2. Die rote, gelbe und grüne LED müssen in 2-Sekunden Intervallen fließend nacheinander ein- und ausgeschaltet werden. Das fließende Licht wird erzeugt, in dem man die LED mit mehreren hundert Hertz moduliert (also sehr schnell ein- und ausschaltet) und die Abstände zwischen dem Ein- und Ausschalten verändert. Dadurch wird für das Auge der Effekt erzeugt, dass sich die Helligkeit der LED ändert (Pulsweitenmodulation).

3. Erstellen Sie einen Programmteil in C, welcher eine Verzögerung von 10 μs $\pm$ 10% bei einer Taktfrequenz von 14,7456 MHz erzeugt. Verifizieren Sie theoretisch, dass die Verzögerung funktioniert, indem Sie die Assembler-Befehle in der .lss Datei des kompilierten Programms kommentieren.

Fragen

1. Welche Methoden zur Erstellung von Verzögerungen gibt es?

2. Wie wird eine Softwareverzögerung erstellt? Von welchen Parametern hängt die Softwareverzögerung ab?

3. Warum werden Hardwareverzögerungen/Timer mit Interrupts genutzt?

4. Berechnen sie die die Überlauf-Interruptperiode für einen 8-Bit Timer, wenn Taktfrequenz 16M Hz und Frequenzteilerfaktor 1024 betragen.

5. Was ist ein Echtzeitgeber in einem Computer?

6. Was geschieht am 19.01.2038 in der Computerwelt?

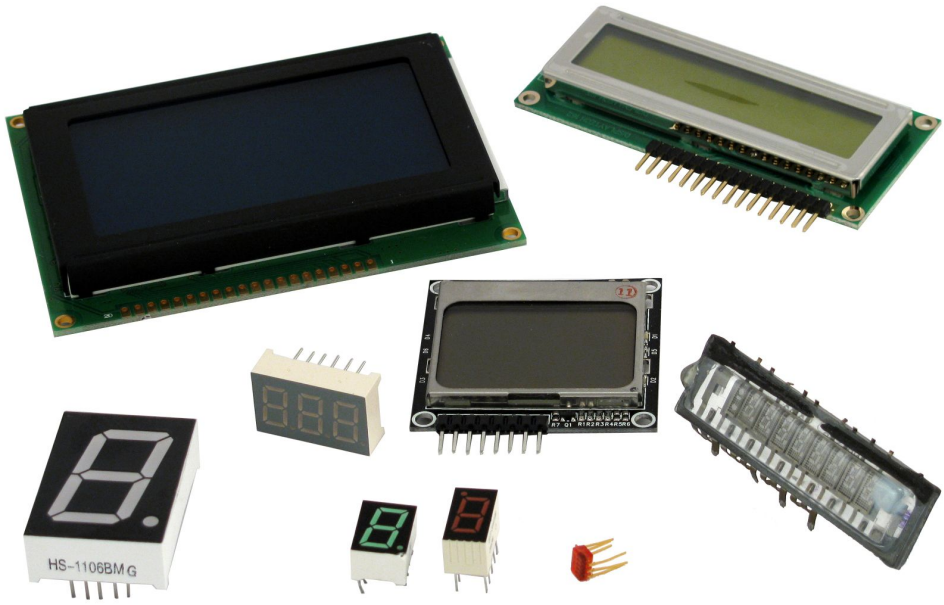
7. Wozu können AVR Timer noch genutzt werden, außer um Zeit zu zählen?

8. Welche Indizes können genutzt werden um den ATmega128 Timer 0 einzustellen? Was kann mit diesen Register verändert werden?

9. Wie lang ist die längste Zeitspanne von Unterbrechungen in Millisekunden, die mit einem ATmega128 mit einer Taktfrequenz von 14,7456 MHz erreicht werden kann? Stellen Sie die Berechnung dar.

10. Hat es einen Effekt auf die Genauigkeit von Timern, wenn der Prozessor mit der Ausführung eines Programms stark ausgelastet ist (z.B. mehrere Motoren und Werte verschiedener Sensoren gleichzeitig kontrollieren)? Erläutern Sie Ihre Antwort.

6.4 Anzeigen und Displays



Einer der einfachsten Methoden um sicher zu gehen, dass ein Mikrocontroller funktioniert ist Code zu einem Programm hinzuzufügen, um eine oder mehrere LEDs bei einer bestimmten Aktivität blinken zu lassen. Sehr oft ist dies nicht genug, weil mehr Informationen überprüft werden müssen, als mit einer LED möglich oder da einige Anwendungen brauchen eine Benutzerschnittstelle, welche Text oder Nummern anzeigt, benötigen. Hier finden alle Arten von Indikatoren und Displays Verwendung, welche das Anzeigen von Nummern, Texten oder sogar Bildern ermöglichen. Computerbildschirme zählen zu den bekanntesten Anzeigegeräten. Für einen Programmierer ist die Arbeit mit ihnen alltäglich, jedoch ist es sehr aufwändig Pixel mit Hilfe von Mikrocontrollern auf einem Bildschirm anzuzeigen.

Das folgende Kapitel befasst sich mit einfachen Displays und Bildschirmen. Anzeigen mit LED-Segmenten und zwei Typen von monochromen LCDś. Zusätzlich zu diesen werden auch LED-Matrizen und organischen Farb-LEDś(OLED) mit Mikrocontrollern genutzt. Früher wurden Nixie-Röhren verwendet, welche separate Glühdrähte für jede Zahl hatten. Darüber hinaus gibt es elektromechanische Anzeigen, deren verschiedenfarbige Segmente physisch gedreht werden. Neuerdings erfreuen sich E-Paper-Displays, mit denen gedruckte Texte nachgebildet werden können, großer Beliebtheit. Die Ersten E-Paper (elektronisches Papier) bestanden aus Kugeln mit verschieden farbigen Seiten, welche in einem elektrischen Feld gedreht wurden. Neuere E-Paper enthalten kleine Kapseln, in denen geladene Partikel zweier Farben durch ein elektrisches Feld dahingehend beeinflusst werden, ob sie herabsinken oder an der Oberfläche bleiben sollen. Die letzten beiden Beispiele wurden zur Erläuterung diverser Anzeigetechniken dargestellt, sie werden im HomeLab jedoch nicht

weiter verfolgt.

6.4.1 7-Segment-LED-Anzeige

Notwendiges Wissen: *User Interface Module,* *Delay,* *7-segment LED Display,* *Light-emitting Diode*

Theorie

Die 7-Segment-LED-Zifferanzeige ist ein Display, welches aus sieben, in Form einer Acht angeordneten, LEDs besteht. Durch Ein- oder Ausschalten der entsprechenden LEDś (Segmente) ist es möglich Zahlen von 0-9 sowie einige Buchstaben darzustellen.

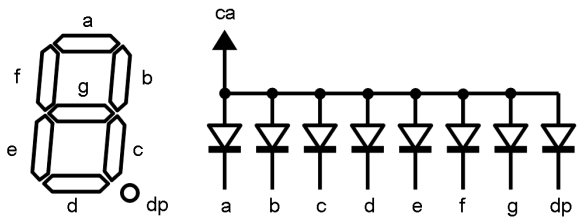


Abbildung 6.11: Positionierung der Anzeigensegmente des LED sowie elektrisches Schema.

Elektrisch werden alle Anoden der LEDs an einer Anode am Pin *ca* angeschlossen. Die LEDs leuchten durch Schalten ihrer Kathoden (*a, b, c . . .*). Es gibt auch andere Verbindungen, bei denen die Anzeigen eine gemeinsame Kathode *cc* haben. Allgemein werden verschiedene Zifferanzeigen zur Darstellung mehrstelliger Ziffern genutzt. Hierzu verfügt das Display über ein Komma- (Punkt-) Segment *dp*. Insgesamt besitzt eine Anzeige dann 8 Segmente, wird jedoch aufgrund der Anzahl von Ziffer-Segmenten weiterhin 7-Segment-Display genannt.

LED number-indicators are easy to use, they can be controlled directly from the pins of the microcontroller, but there are also special drivers, which able to control number-indicators using fewer pins of the microcontroller. There are different colors of LED number indicators, which can be very bright and very large. For displaying the entire Latin alphabet exist indicators with extra segments.

Die Benutzung von LED Zifferanzeigen ist recht einfach. Sie können direkt über die Pins des Mikrocontrollers kontrolliert werden. Es gibt jedoch auch spezielle Treiber, die dafür sorgen, dass für die Nutzung der Anzeige weniger Pins des Mikrocontrollers benötigt werden. Es gibt zwei verschiedene Farben von LED-Zifferanzeigen, die sehr klar und groß sein können. Zur Anzeige des gesamten Alphabets gibt es Anzeigen mit zusätzlichen Segmenten.

Übung

Am digitalen I/O Modul befindet sich eine 7-Segment-LED-Zifferanzeige. Sie wird über einen Treiber mit der seriellen Schnittstelle A6275 kontrolliert. Die serielle Schnittstelle des Treibers ist dem SPI ähnlich, bei dem sowohl Taktsignal als auch Datensignale genutzt werden. Anders als beim SPI wird *chip-select* nicht genutzt und wird durch die *latch* Funktion ersetzt. Es ist mit dem ATmega128 wie folgt verbunden:

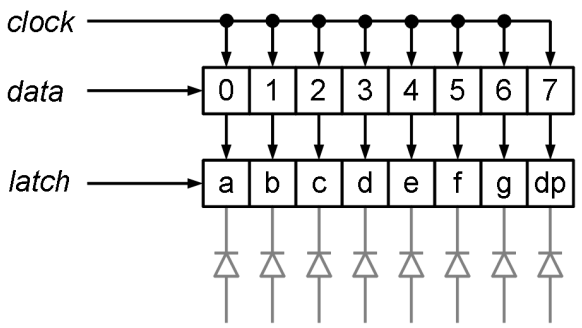


Abbildung 6.12: Aufbau des Verschiebungsindexes des LED Treibers mit den zugehörigen Segmenten der Anzeige.

Latch-Signal (*latch*) - PG2

Taktsignal (*clock*) - PC7

Datensignal (*data*) - PC6

Die Daten werden in Form von Bits über den Datenpin gesendet. Immer, wenn das Taktsignal high wird, wird der Inhalt des Verschiebungsindexes nach rechts geschoben und das Bit vom Datenpin wird in die Zelle ganz links gelesen. So werden 8 Bits in den Verschiebungsindex geladen. Wenn das Latch-Signal high gesetzt wird, wird der Wert des Verschiebungsindexes in den Latch-Index geladen, und verbleibt dort bis zu einem erneuten Laden. Jedes Bit des Latch-Indexes ist durch eine Spannungsschaltung mit einem Segment der Zifferanzeige verbunden. Das Segment leuchtet während die Bitrate high ist.

Zur Darstellung von Ziffern auf der digitalen I/O Modulanzeige des HomeLab sind die folgenden Funktionen in der Bibliothek von HomeLab enthalten:

```
//
// Einrichtung der Pins
//
static pin segment_display_latch =    PIN(G, 2);
static pin segment_display_data_out = PIN(C, 6);
static pin segment_display_clock =    PIN(C, 7);

//
// Markierung der Karte.
```

```
// Die Bits markieren Segmente. Ein niedriger Rang wird mit A, ein
    hoher mit DP gekennzeichnet.
//
static const unsigned char segment_char_map[11] =
{
  0b00111111,
  0b00000110,
  0b01011011,
  0b01001111,
  0b01100110,
  0b01101101,
  0b01111100,
  0b00000111,
  0b01111111,
  0b01100111,
  0b01111001    // E wie "Error"
};

//
// Start der 7-Segment-Anzeige.
//
void segment_display_init(void)
{
  // Set latch, data out and clock pins as output
  pin_setup_output(segment_display_latch);
  pin_setup_output(segment_display_data_out);
  pin_setup_output(segment_display_clock);
}

//
// Darstellung einer Ziffer auf der 7-Segment-Anzeige.
//
void segment_display_write(unsigned char digit)
{
  unsigned char map;
  signed char i;

  // Überprüfung der Ziffer
  if (digit > 9)
  {
    digit = 10;
  }

  // Ziffer wie Segmentkarte.
  map = segment_char_map[digit];
```

```
  // Latch-Signal aus
  pin_clear(segment_display_latch);

  // Senden von he Bits. Höherrangige zuerst.
  for (i = 7; i >= 0; i--)
  {
    // Pin entsprechend Bitwert der Karte setzen.
    pin_set_to(segment_display_data_out, bit_is_set(map, i));

    // Taktsignal high setzen.
    pin_set(segment_display_clock);
    _delay_us(1);

    // Taktsignal low setzen.
    pin_clear(segment_display_clock);
    _delay_us(1);
  }

  // Latch-Signal ein.
  pin_set(segment_display_latch);
}
```

Um Ziffern und den Buchstaben „E“darzustellen, wird ein konstantes Array *segment_char_map* erstellt, bei dem das Leuchten von allen acht Segmenten mit Bit 1 und das Ausschalten mit Bit 0 markiert wird. Die Bits markieren von hoch zu niedrig (von links nach rechts in Binärform) Segmente DP, G, F, E, D, C, B, A. Die Kontrollschnittstelle des Treibers wird durch SPI Software realisiert, zum Beispiel durch Nutzung einer Software, die die Datenkommunikationspins des Programms kontrolliert. Alle drei Pins werden durch *segment_display_init* als Outputs gesetzt. *segment_display_write* dient der Darstellung der Funktion, welche die Segmentkarte der Markierung vom Array erkennt und diese bitweise zum Treiber überträgt. Die Taktfrequenz mit Softwareverzögerung liegt bei ungefähr 500 kHz.

Das nachfolgende Beispielprogramm beschreibt die Nutzung der Zifferanzeige. Die zuvor beschriebene Funktion der Bibliothek wird in dem Programm erläutert. Das Programm Zählt Ziffern von 0 bis 9 mit einem Intervall von etwa einer Sekunde. Dieses wird dadurch erreicht, dass ein Modul mit einer weitaus größeren Ziffer verwendet wird.

```
//
// Das Beispielprogramm der 7-Segment-LED-Anzeige des I/O-Moduls des
    HomeLab
//
#include <homelab/module/segment_display.h>
#include <homelab/delay.h>

//
// Hauptprogramm.
//
```

```
int main(void)
{
  int counter = 0;

  // Einrichtung der 7-Segment-Anzeige.
  segment_display_init();

  // Endlosschleife.
  while (true)
  {
    // Anzeige der Werte des Timers.
    segment_display_write(counter % 10);

    // Lange zählen.
    counter++;

    // Verzögerung von 1 Sekunde.
    sw_delay_ms(1000);
  }
}
```

6.4.2 Alphanumerisches LCD

Notwendiges Wissen: *lcd,* *Alphanumeric LCD,* *Delay,* *Periodic interrupt*

Theorie

Das alphanumerische LCD ist ein Flüssigkristall-Display, welches zur Darstellung von Buchstaben und Ziffern dient. In einfachen LCD's werden flüssige Kristalle verwendet, welche zwischen transparenten Elektroden angeordnet sind und die Polarisation des durchstrahlenden Lichts im elektrischen Feld verändern. Auf den Elektroden befinden sich Polarisationsfilter, welche sicher stellen, das nur in eine Richtung polarisiertes Licht auf den Bildschirm trifft.Wenn der Flüssigkristall seine Polarisation durch das elektrische Feld ändert, erreicht das Licht den Bildschirm oder ein Segment des Bildschirms nicht und er bleibt schwarz.

Hauptmerkmal eines alphanumerischen LCD ist die Anordnung der Segmente. Der Bildschirm ist in verschiedene Anzeigen unterteilt. Jede Anzeige hat entweder genügend Segmente um Buchstaben oder Nummern anzuzeigen oder es wird eine Matrix aus Pixeln gebildet. Eine Matrix aus 5×7 Pixeln ist beispielsweise ausreichend, um alle Nummern und Buchstaben des lateinischen Alphabets darzustellen. Die Anzeigeelemente bestehen gewöhnlich aus 1 bis 4 Zeilen sowie 8 bis 32 Spalten. Zwischen den Elementen befinden sich kleine Abstände ähnlich wie bei Buchstaben im Text.

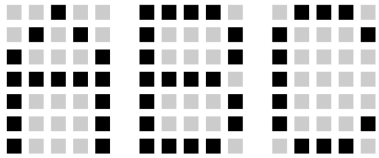


Abbildung 6.13: Darstellung eines Textes durch eine Pixelmatrix eines alphanumerischen LCD.

Neben dem Bildschirm hat das alphanumerische LCD auch einen Controller, welcher die Segmente des Bildschirms gemäß den Befehlen der Kommunikationsschnittstelle steuert. Ein Controller verfügt über eine vorprogrammiertes Set von Buchstaben mit einem eigenen Index für jeden Buchstaben, jede Nummer sowie jedes Symbol. Die Darstellung des Textes auf dem Bildschirm erfolgt, einfach gesagt, indem die Indizes zum Controller gesendet werden. Tatsächlich müssen jedoch weitere Kontrollbefehle übermittelt werden, bevor irgendetwas dargestellt werden kann. Da es viele verschiedene Ausführungen von LCDs gibt, die jeweils unterschiedlich gesteuert werden, ist es wichtig, sich mit dem Datenblatt jedes LCDs vertraut zu machen.

Alphanumerische LCDs besitzen normalerweise eine passive Matrix. Hier erfolgt die Erneuerung des elektrischen Feldes der Segmente turnusmäßig. Aus diesem Grund sind Bildschirme mit einer passiven Matrix langsamer und Darum sind die Bildschirme mit passiver Matrix langsamer und bieten weniger Kontrast, als Bildschrime mit aktiver Matrix. Bei Aktivmatrix-Bildschrimen wird die Ladung jedes einzelnen Segments durch einen eigenen Transistor gesteuert. Einige LCDs haben einen reflektierenden Hintergrund, andere haben Hintergrundbeleuchtungen und wieder andere sogar verschiedene Hintergrundbeleuchtungen. Die Segmente eines alphanumerischen LCDs haben jedoch immer nur eine Farbe. In der Regel sind sie schwarz, es gibt jedoch auch Bildschrime mit weißer oder farbiger Schrift.

Übung

Am digitalen I/O Modul des HomeLab ist ein alphanumerisches, 2×16 Zeichen großes, LCD WC 1602A angeschlossen. Zur Steuerung des Bildschirms gibt es einen 4-Bit Datenbus sowie 3 Kontroll-Pins. Da das Kommunikationsprotokoll des Bildschirms zu umfangreich ist, um es hier zu erläutern, enthält die HomeLab Library die entsprechenden Funktionen zur Nutzung des Displays.

Vor der Nutzung des Displays ist es notwendig, seine Einstellungen anzupassen. Hierzu dient die *lcd_alpha_init* Funktion, welche einen blinkenden Cursor auf dem Bildschirm erscheinen lässt. Hier gibt es nur eine aktive Position für den Cursor, an welcher der nächste Buchstabe eingegeben werden kann. Bevor also Text eingegeben wird, muss der Cursor an die gewünschte Position bewegt werden. Um die Position des Cursors zu verändern gibt es die Funktion *lcd_alpha_goto_xy* und für die Darstellung die Funktion *lcd_alpha_write_string*. Sämtliche Funktionen des alphanumerischen LCDs sind in der Library erklärt.

Der folgende Programm-Code zeigt wie das alphanumerische LCD als Uhr genutzt werden kann. Die Zeit beginnt bei 00:00:00 und nimmt etwa jede Sekunde zu. Da das Zählen der Zeit mit Hilfe einer Verzögerungsfunktion geschieht, ist das Ergebnis nicht sehr genau. Diese Ungenauigkeit wird im Rahmen der Aufgabe zu periodischen Unterbrechungen erklärt. Das Programm zählt die Sekunden und konvertiert diese in Minuten und Sekunden. Um Uhrzeit darzustellen wird die Standard-Funktion in C genutzt: *sprintf*.

```
//
// Beispiel zur Nutzung des alphanumerischen LCD des HomeLab.
```

```
// Die Startzeit zu Beginn des Programms wird auf dem LCD angezeigt.
//
#include <stdio.h>
#include <homelab/module/lcd_alpha.h>
#include <homelab/delay.h>

//
// Hauptprogramm.
//
int main(void)
{
    int seconds = 0;
    char text[16];

    // Einrichtung des LCD.
    lcd_alpha_init(LCD_ALPHA_DISP_ON);

    // Löschen des LCD.
    lcd_alpha_clear();

    // Name des Programms.
    lcd_alpha_write_string("The Time Counter");

    // Endlosschleife
    while (true)
    {
        // Konvertieren der Sekunden in die Zeitanzeige:
        // hh:mm:ss
        sprintf(text, "%02d:%02d:%02d",
            (seconds / 3600) % 24,
            (seconds / 60) % 60,
             seconds % 60);

        // Anzeige der Zeit am Anfang der zweiten Reihe des LCD.
        lcd_alpha_goto_xy(0, 1);
        lcd_alpha_write_string(text);

        // Anstieg der Sekunden um 1.
        seconds++;

        // Hardwareverzögerung 1000 ms.
        hw_delay_ms(1000);
    }
}
```

6.4.3 Graphisches LCD

Notwendiges Wissen: *lcd,* *Graphic LCD,* *Delay,* *Alphanumeric LCD*

Theorie

Das graphische LCD *liquid crystal display* ist ein Display zur Darstellung von Bildern und Text. Es ist ähnlich aufgebaut wie das alphanumerische LCD, mit dem Unterschied, dass sich auf dem graphischen LCD alle Pixel in einer großen Matrix befinden. Bei monochromen LCDs entspricht ein Pixel einem quadratischen Segment, bei Farbdisplays besteht ein Pixel aus drei Unterpixeln. Jedes der Unterpixel lässt nur eine Lichtfarbe durch (rot, grün oder blau). Da die Unterpixel sehr nah beieinander liegen, werden sie als ein Pixel wahrgenommen.

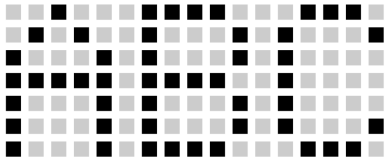


Abbildung 6.14: Der durch Pixel eines graphischen LCDs dargestellte Text.

Monochrome graphische Displays haben normalerweise eine passive Matrix, große Farbdisplays (auch Computer-Bildschirme) eine aktive. Im Hinblick auf die Farbe des Hintergrunds und der Pixel entspricht das graphische LCD dem alphanumerischen. Auch das graphische Display verfügt über einen separaten Controller, welcher die Information über die Kommunikationsschnittstelle empfängt und das elektrische Feld für die Segmente generiert. Im Gegensatz zum alphanumerischen LCD, welches Text anzeigt, indem die Indizes der darzustellenden Zeichen gesendet werden, kann das graphische LCD keine Buchstaben eigenständig generieren - sämtliche Bilder und Texte müssen Pixel für Pixel vom Nutzer generiert werden.

Übung

Das HomeLab Kit enthält ein 84×48 Pixel großes, monochromes, graphisches LCD. Es ist das gleiche, welches auch im Mobiltelefon Nokia 3310 verwendet wird. Ein Philips PCD8544 Controller ist am Display angebracht, mit welchem über eine SPI-ähnliche serielle Schnittstelle kommuniziert werden kann. Die Hintergrundbeleuchtung des Displays wird separat gesteuert.Die Kommunikation mit dem Display ist nicht sehr schwer, aufgrund der vielen Funktionen wird sie hier jedoch nicht erläutert. Die HomeLab Library enthält die dafür notwendigen Funktionen.

Die Funktionen für das graphische LCD sind ähnlich zu denen des alphanumerischen LCD. Zuerst muss der Bildschirm mit der Funktion *lcd_gfx_init*gestartet werden. Nach dem Start sollte man den Bildschrim, bzw. den Speicher des Controllers mit der *lcd_gfx_clear* Funktion leeren. Die Library enthält eine Abbildung des vollständigen lateinischen Alphabets, der Zahlen sowie der am häufigsten verwendeten Zeichen. Die Buchstaben ist 7 Pixel hoch und 5 Pixel breit. Der Abstand zwischen den Buchstaben beträgt horizontal 6 und vertikal 8 Pixel. Es können also Buchstaben in 6 Reihen mit 14 Spalten auf dem Display dargestellt werden. Zur Darstellung von

Buchstaben oder Text, muss zunächst mit der Funktion *lcd_gfx_goto_char_xy* die Position festgelegt werden. Mit der Funktion *cd_gfx_write_char* wird ein Buchstabe angezeigt, die Funktion *lcd_gfx_write_string* ermöglicht die Darstellung eines Textes.

Nachfolgend ist ein Beispiel des Zeitzählers dargestellt. Das Programm zählt Sekunden (annähernd), Minuten und Stunden. Zum Konvertieren von Zeit in Text wird die Funktion *sprintf* genutzt.

```
//
// Beispiel für die Nutzung des graphischen LCD des HomeLab.
// Die Tageszeit wird ab Beginn des Programms auf dem LCD angezeigt.
//
#include <stdio.h>
#include <homelab/module/lcd_gfx.h>
#include <homelab/delay.h>

//
// Hauptprogramm.
//
int main(void)
{
    int seconds = 0;
    char text[16];

    // Einrichten des LCD.
    lcd_gfx_init();

    // Löschen des Displays.
    lcd_gfx_clear();

    // Einschalten der Hintergrundbeleuchtung.
    lcd_gfx_backlight(true);

    // Anzeige des Programmnamens.
    lcd_gfx_goto_char_xy(1, 1);
    lcd_gfx_write_string("Aja loendur");

    // Endlosschleife.
    while (true)
    {
        // Sekunden in die Form der Uhrzeit konvertieren.
        // hh:mm:ss
        sprintf(text, "%02d:%02d:%02d",
            (seconds / 3600) % 24,
            (seconds / 60) % 60,
             seconds % 60);

        // Anzeige des Zeittextes.
```

```
        lcd_gfx_goto_char_xy(3, 3);
        lcd_gfx_write_string(text);

        // Eine Sekunde hinzufügen.
        seconds++;

        // Hardwareverzögerung von 1000 ms.
        hw_delay_ms(1000);
    }
}
```

6.4.4 Aufgaben

Ziel ist es, ein Programm zu schreiben, welches die unten beschriebenen Aufgaben erüllt.

Aufwärmübung

Das Programm soll von 1 bis 9 und dann zurück von 9 bis 1 auf der 7-Segment-Anzeige zählen. Die Zählperiode beträgt 1 Sekunde.

Für Anfänger

1. Stellen Sie die Zahlen im Hexadezimalsystem zufällig auf der 7-Segment-Anzeige dar. Die Frequenz beträgt 1 Hz.

2. Lassen Sie die äußeren 6 Segmente der 7-Segment-Anzeige periodisch im Abstand von 500 ms aufleuchten.

3. Schreiben Sie ein Programm für das LCD, welches den Code und das zugehörige Symbol anzeigt. Verwenden Sie Codes von 0 bis 255. Jedes Code-Symbol-Paar wird 1 Sekunde lang auf dem Display angezeigt.

4. Schreiben Sie ein Programm, mit welchem das Symbol „X“über den Bildschrim bewegt werden kann. Mit dem Schalter S1 kann es nach links bewegt werden, mit Schalter S3 nach rechts und mit Schalter S2 wird die Reihe gewechselt.

5. Stellen Sie auf dem graphischen Display 10 Zeilen Text dar. Mit den Schaltern S1 und S2 kann innerhalb des Textes hoch- und runtergescrollt werden.

6. Erstellen Sie eine Benutzerschnittstelle über welche mit den drei Schaltern Text eingegeben werden kann. Ein Schalter wählt beispielsweise das Zeichen aus, der zweite überprüft das Zeichen und der dritte überprüft schließlich den Text. Die maximale Länge des Textes beträgt 10 Zeichen und der Text

soll in umgekehrter Reihenfolge in der zweiten Reihe dargestellt werden. Sie können das lateinische Alphabet nutzen und das Display selbst auswählen.

Für Fortgeschrittene

1. Stellen Sie griechische Buchstaben auf dem graphischen LCD dar. Dabei sollen folgende Zeilen angezeigt werden: "Widerstand Ω", "∑R=∑ πR^2", "π=3.141592". Nutzen Sie den Quellcode aus der HomeLab Library (von der Webseite).

2. Schreiben Sie ein Programm, um Dezimalzahlen in Binärzahlen zu konvertieren. Verwenden Sie die Schalter S3 bis S1 zum Einfügen von Dezimalzahlen in zehn Sekunden (S3 - Hunderter S2 - Zehner S1 - Einer). Durch vierfaches Betätigen von Schalter S3 wird so beispielsweise die Zahl 400 angezeigt. Das Zeichen für den Beginn der Eingabe und für das Ergebnis sollen in Binärzahlen ausgegeben werden. Verwenden Sie hierzu ein beliebiges LCD.

3. Schreiben Sie eine Funktion zur Darstellung eines Rechtecks auf dem LCD. Breite, Länge sowie die Koordinaten der oberen linken Ecke sind dabe vorgegeben. Die Linienstärke beträgt 1 Pixel. Überprüfen Sie, ob das Rechteck ins Display passt. Hierzu ist es ratsam, sich mit der HomeLab Library zu befassen.

4. Programmieren Sie ein einfaches Wurm-Spiel für das graphische LCD. Der Wurm ist 1 Pixel breit und 5 Pixel lang. Durch Betätigung der Schalter kann der Wurm sich nach rechts oder links bewegen. Er muss auf diese Weise Kollisionen mit dem Displayrand vermeiden können. Es gibt Bonuspunkte für das Einsammeln von Eiern, wodurch der Wurm wächst. Die einfachste Lösung ist, den Wurm aus dem Buchstaben „O"zu entwickeln.

5. Schreiben Sie ein Programm zur Darstellung verschiedener Wetterbedingungen auf dem graphischen LCD. Nutzen Sie für sonniges Wetter ein Bild der Sonne, für Regen eine Regenwolke, für bewölkt eine Wolke und für Schnee eine Schneeflocke. Die Größe der Bilder kann variieren, wichtig ist nur, dass die Bilder sich deutlich voneinander unterscheiden. Die Bilder sollen mittels eines Schalters gewechselt werden können.

Fragen

1. Wie viele Pins nutzt die 7-Segment Zifferanzeige (mit Punktsegment), wenn sie direkt mit dem Controller verbunden ist? Wie viele Pins werden benötigt, wenn sie über einen Treiber gesteuert wird? (*Treiber* - A6275)

2. Was bestimmt die Helligkeit der 7-Segment Zifferanzeige? Wie kann sie angepasst werden wenn die Anzeige a) direkt oder b) über einen Treiber (*Treiber* - A6275) gesteuert wird?

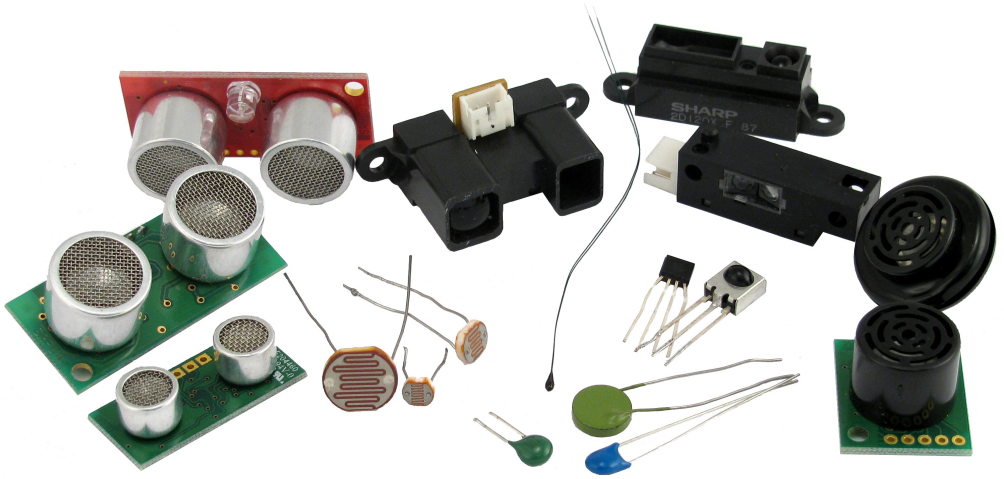
3. Wenn die 7-Segment Zifferanzeige direkt an Port A des Controllers angeschlossen ist sodass das Segment A PA0, B PA1 ... und DP PA7 ist, wie lauten dann die Werte des PORTA Registers mit den Ziffern 0 bis 9?

4. Was ist der Unterschied zwischen alphabetischen 4-Bit und 8-Bit LCD Controllern?

5. Über welche Pins und wie wird die Hintergrundbeleuchtung der alphanumerischen LCDs reguliert?

6. Welches I/O Protokoll nutzt das graphische LCD? Erläutern Sie die Bedeutung der I/O Pins.
7. Wie können Dezimalzahlen in binäre (in Text) konvertiert werden und umgekehrt?
8. Zeichnen Sie die Schichten aus denen ein LCD besteht mittels der *twisted nematic* Technologie.
9. Wie werden Buchstaben auf dem graphischen LCD angezeigt?
10. Wie unterscheidet sich ein monochromes (schwarz/weiß) LCD von einem Farbdisplay?

6.5 Sensoren



Sensoren sind Geräte welche jede Form von physikalischen Attributen (Temperatur, Beleuchtung, Kraft, Beschleunigung usw.) in eine verständliche Form für Menschen oder Maschinen bringen. Mit der Hilfe von Sensoren erhält der Mikrocontroller Informationen über seine Umgebung und trifft Entscheidungen basierend auf diesen Informationen. Es gibt viele Arten von Sensoren, ca. 195 verschiedene Typen werden in Wikipedia aufgelistet. Nur Sensoren mit einem elektrischen Outputsignal können an einem Mikrocontroller angeschlossen werden. Basierend auf dem elektrischen Outputsignal ist es möglich, Sensoren in digitale oder analoge Sensoren zu unterteilen.

Bei analogen Sensoren bewirkt jede physikalische Änderung eine Änderung des elektrischen Wertes, i. d. R. Spannung, Stromstärke oder Widerstand. Da Mikrocontroller digitale Geräte sind, muss das analoge Signal in ein digitales umgewandelt werden, bevor es zum Controller gesendet wird. Dafür werden Analog-zu-Digital Konverter benutzt, welche normalerweise im Mikrocontroller verbaut sind.

Analoge Sensoren welche die Information digitalisieren, werden digitale Sensoren genannt. Digitale Sensoren

können darüber hinaus Information standardisieren, Sensoren kalibrieren und viele andere Funktionen ausführen. Es gibt verschiedene Möglichkeiten um Information von einem digitalen Sensor an einen Mikrocontroller zu senden. Die einfachste Möglichkeit ist es mit logischen Signalen Informationen zu übertragen. Eine komplexere Variante geht über eine Datalink Schnittstelle. Die folgenden Übungen führen aber nur einfachere und in der Robotik genutzte Sensoren ein.

6.5.1 Potentiometer

Notwendiges Wissen: *Sensors Module,* *User Interface Module,* *Voltage Divider,* *Analog-to-digital Converter,* *Analog to Digital Converter,* *7-segment LED Display*

Theorie

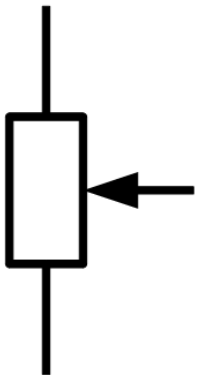


Abbildung 6.15: elektrisches Symbol eines Potentiometers

Ein Potentiometer ist ein elektronisches Widerstandsbauelement mit drei Anschlüssen. Zwischen den beiden Seitenkontakten ist der Widerstand fix, zwischen Seiten- und Mittelkontakt ist er variabel. Im Grunde ist ein Potentiometer ein Spannungsteiler, dessen Widerstand zwischen den Seiten- und Mittelkontakten hergestellt wird.

Abbildung 6.16: einfaches Drehpotentiometer

Ein typisches Potentiometer besteht aus einem Widerstand mit einer leitenden Oberfläche und einem Schiebekontakt (slider). Je näher der Schiebekontakt am Rand des Widerstandes angebracht ist, desto geringer ist der Widerstand zwischen Slider und Ecke des Widerstands und umgekehrt. Ein Material mit hohem Widerstand oder einer Spule aus Widerstandsdraht kann als Widerstand fungieren. Bei einigen Potentiometern sind die Beziehungen zwischen Widerstand und Sliderposition linear oder logarithmisch. Potentiometer sind normalerweise einfache Drehpotentiometer (siehe Bild), es gibt aber auch Schiebepotentiometer. Ein spezieller Typ von Potentiometer sind digitale Potentiometer, bei denen die Regulation des Widerstands intern über Signale geschieht.

Übung

Am HomeLab Modul befindet sich ein 4,7 kΩ Drehpotentiometer. Das Potentiometer ist mit der Masse und dem +5 V Potential angeschlossen, und der Schiebekontakt ist am Kanal 3 des analog-digital Konverters angeschlossen. So kann der Spannungsoutput des Potentiometers zwischen 0 und 5 V reguliert werden. Der digitale Wert der Potentiometeroutputspannung kann gemessen werden indem die Vergleichsspannung vom AVR digital-analog Konverter vom AVCC Pin genommen wird. die Folgende Funktionen für den ACR ADC sind in der HomeLab Bibliothek enthalten.

```
//
// Data types for adjustment
//
typedef enum
{
  ADC_REF_AREF = 0x00,
  ADC_REF_AVCC = 0x01,
  ADC_REF_2V56 = 0x03
}
adc_reference;

typedef enum
{
  ADC_PRESCALE_2    = 0x01,
  ADC_PRESCALE_4    = 0x02,
  ADC_PRESCALE_8    = 0x03,
```

```
  ADC_PRESCALE_16  = 0x04,
  ADC_PRESCALE_32  = 0x05,
  ADC_PRESCALE_64  = 0x06,
  ADC_PRESCALE_128 = 0x07
}
adc_prescale;

//
// Starting the ADC
//
void adc_init(adc_reference reference, adc_prescale prescale)
{
  // Allowing ADC to operate, selecting the frequency divider
  ADCSRA = bit_mask(ADEN) | (prescale & 0x07);

  // Selecting comparison voltage
  ADMUX = (reference & 0x03) << REFS0;
}

//
// Converting the values of selected channel
//
unsigned short adc_get_value(unsigned char channel)
{
  // Setting the channel
  ADMUX = (ADMUX & 0xF0) | (channel & 0x0F);

  // Starting the conversion
  bit_set(ADCSRA, ADSC);

  // Waiting the end of the conversion
  while (bit_is_set(ADCSRA, ADSC))
  {
    asm volatile ("nop");
    }

  // Returning the results
  return ADCW;
}
```

Die Funktion *adc_init* muss zu Beginn des Programms ausgeführt werden. Sie sorgt dafür, dass der ADC funktioniert. Die Vergleichsspannung muss entweder vom AREF oder AVCC Pin kommen, oder es muss die eingestellte interne Spannung von 2,56 V ausgewählt werden. Dazu muss der Taktzyklus des Konverters mit einem Vorzähler gesetzt werden (Faktor des Frequenzteilers), welcher den Taktzyklus des Controllers teilt. Die Umsetzung ist schneller, wenn höhere Taktzyklen verwendet werden, jedoch kann die Genauigkeit drunter leiden. Die Funktion *adc_get_value* dient zur Messung. Es kann der Kanal gewählt werden und sie gibt 10-Bit

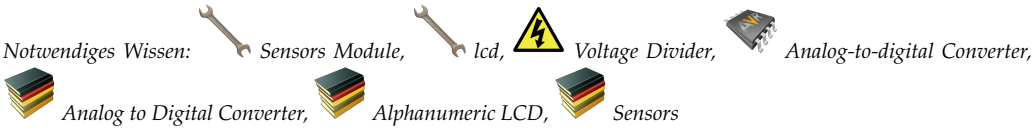
Ergebnisse als ganze 16-Bit Zahlen aus. Die Funktion zur Messung wartet bis die Konversion beendet ist und gibt das Ergebnis erst dann aus.

In den zuvor erklärten Beispielprogrammen werden die Bibliotheken des ADC und des 7-Segment-Zifferanzeige genutzt. Der 10-Bit Wert des ADC wird mit 10 multipliziert und durch 1024 dividiert um einen Wert zwischen 0 und 9 zu erhalten. Der Wert 10 kann nicht erreicht werden, weil in C nur ganzzahligewerte berechnet werden und keine gerundeten Ergebnisse.Um die Genauigkeit des Ergebnisses zu erhöhen, wird eine Funktion zur Berechnung des Durchschnitts der Ergebnisse des ACSs genutzt. Daraus abgeleitet gibt das Programm auf der Anzeige einen Wert von 0 bis 9 aus, welcher der Position des Potentiometers entspricht.

```
//
// Beispielprogramm für das Potentiometer des Sensormoduls
// Die Position des Potentiometers wird auf der 7-Segmentanzeige
    dargestellt
//
#include <homelab/adc.h>
#include <homelab/module/segment_display.h>

//
// Auswahl des Kanals
//
//  1 = Photoresistor
//  2 = Thermistor
//  3 = Potentiometer
//
#define ADC_CHANNEL 3

//
// Hauptprogramm
//
int main(void)
{
  int value;

  // Anpassung der 7-Segmentanzeige
  segment_display_init();

  // Anpassung des ADC
  adc_init(ADC_REF_AVCC, ADC_PRESCALE_8);

  // Endlosschleife
  while (true)
  {
    // Liest viermal gerundete Werte aus dem Kanal
    value = adc_get_average_value(ADC_CHANNEL, 4);

    // Zeigt Hunderter des angezeigten Wertes an
```

```
    segment_display_write(value * 10 / 1024);
  }
}
```

6.5.2 Thermistor



Theorie

Abbildung 6.17: NTC Thermistor

Ein Thermistor ist temperaturabhängiger Widerstand. Es gibt Thermistoren mit positiven und mit negativen Temperaturkoeffizienten. Der Widerstand von Thermistoren mit positiven Koeffizienten nimmt mit steigender Temperatur zu, bei Thermistoren mit negativen Koeffizienten steigt er mit sinkender Temperatur. Die dazugehörigen Abkürzungen sind PTC(*positive temperature coefficient*) und NTC (*negative temperature coefficient*).

Die Temperaturabhängigkeit des Widerstands verläuft nicht linear zu der Temperatur, was die Nutzung erschwert. Für genaue Temperaturmessungen bei größeren Temperaturschwankungen wird die exponentielle Steinhart-Hart Gleichung dritter Ordnung genutzt, da der Thermistorwiderstand nur innerhalb kleiner Temperaturbereiche linear ist. Für NTC Thermistoren gibts es folgende vereinfachte Steinhart-Hart-Gleichung mit dem Parameter B:

$$R = R_0 \cdot \mathrm{e}^{B \cdot (\frac{1}{T} - \frac{1}{T_0})}$$

mit:

T_0 - nominale Temperatur, normalerweise 25 °C.

R_0 - Widerstand bei nominaler Temperatur.

B - Parameter B.

Der Parameter B ist ein Koeffizient, welcher normalerweise im Datenblatt des Thermistors vorgegeben ist. Aber er ist nur in bestimmten Temperaturbereichen ausreichend konstant, z.B. in Bereichen zwischen 25 und 50 °C oder zwischen 25 und 85 °C. Wenn der gemessene Temperaturbereich größer ist, sollte das Datenblatt des Thermistors zu Rate gezogen werden um die Gleichung zu erhalten.

Normalerweise wird ein Spannungsteiler genutzt, um den Widerstand eines Thermistors zu messen. Dabei wird ein Widerstand durch einen Thermistor ausgetauscht wird und die Inputspannung ist konstant. Es wird die Outputspannung des Spannungsteilers gemessen, welche sich in Abhängigkeit der Widerstandsänderung des Thermistors verändert. Wenn Spannung anliegt, fließt Strom durch den Thermistor, wodurch sich dieser bedingt durch den Thermistorwiderstand aufheizt und damit den Widerstand verändert. Der durch das Aufheizen entstehende Fehler kann berechnet werden, jedoch ist es einfacher einen Thermistor zu nutzen, welcher einen hohen Widerstand hat und sich nicht so viel aufheizt.

Bei begrenzten Ressourcen und geringerem Anspruch auf Genauigkeit, werden zuvor errechnete Diagramme und Tabellen für die Temperaturen genutzt. Normalerweise enthalten die Tabellen Temperaturbereiche und die entsprechenden Werte für Widerstand, Spannung und ADC. Alle exponentiellen Berechnungen wurden bereits durchgeführt sodass der Nutzer nur noch die entsprechende Reihe finden und die Werte auslesen muss.

Übung

Das Sensormodul des HomeLab enthält einen NTC Thermistor mit nominellem Widerstand von 10 kΩ. Bei Temperaturen von 25 bis 50 °C ist der Parameter B des Widerstands 3900. Ein Pin des Thermistors ist an die +5 V Betriebsspannung angeschlossen der andere an Kanal 2 (Pin PF2). Ein typischer 10 kΩ Widerstand ist auch am gleichen Pin des Mikrocontrollers und an die Masse angeschlossen. So entsteht zusammen mit dem Thermistor ein Spannungsteiler. Da hier ein NTC Thermistor genutzt wird, bei welchem der Widerstand sinkt wenn die Temperatur steigt, wird die Outputspannung des Spannungsteilers bei steigender Temperatur höher.

Während der Nutzung des AVR ist es nützlich, eine Tabelle mit den Temperaturwerten und den Werten des ADC zu verwenden, um die korrekte Temperatur zu finden. Es ist sinnvoll, für jede Gradzahl der gewünschten Temperaturstufe des Messbereichs den korrespondierenden ADC Wert aus der Tabelle herauszusuchen,

da die Tabelle aufgrund der 10 Bit ADC Werte sehr groß sein wird. Es wird empfohlen, ein Tabellenkalkulationsprogramm (MS Excel, Openoffice Calc, etc.) zur Erstellung der Tabelle zu nutzen. Die *Steinhart-Hart* Gleichung, welche für den NTC angepasst wurde, gibt den zur entsprechenden Temperatur korrespondierenden Widerstand aus. Abgeleitet aus dem Widerstand, ist es möglich die Outputspannung des Spannungsteilers zu berechnen und daraus den Wert des ADC. Berechnete Werte können wie folgt in das Programm eingefügt werden:

```
//
// Tabelle zur Konvertierung von Temperaturwerten ind ADC Werte.
// Jedes Element des Arrays kennzeichnet ein Grad Celsius.
// Die Elemente beginnen bei -20 Grad und enden bei 100 Grad.
// Ein Array enthält 121 Elemente.
//
const signed short min_temp = -20;
const signed short max_temp = 100;

const unsigned short conversion_table[] =
{
  91,96,102,107,113,119,125,132,139,146,153,
  160,168,176,184,192,201,210,219,228,238,247,
  257,267,277,288,298,309,319,330,341,352,364,
  375,386,398,409,421,432,444,455,467,478,489,
  501,512,523,534,545,556,567,578,588,599,609,
  619,629,639,649,658,667,677,685,694,703,711,
  720,728,736,743,751,758,766,773,780,786,793,
  799,805,811,817,823,829,834,839,844,849,854,
  859,863,868,872,876,880,884,888,892,896,899,
  903,906,909,912,915,918,921,924,927,929,932,
  934,937,939,941,943,945,947,949,951,953,955
};
```

Folgender Algorithmus kann genutzt werden um die mit den ADC Parametern korrespondierende Temperatur zu finden:

```
//
// Konvertierung der ACD Werte in Grad Celsius:
//
signed short thermistor_calculate_celsius(unsigned short adc_value)
{
  signed short celsius;

  // Tabelle von hinten beginnend durchgehen:
  for (celsius = max_temp - min_temp; celsius >= 0; celsius--)
  {
    // Ist der Wert aus der Tabelle gleich oder höher dem gemessenen
    // Wert, ist die Temperatur mindestens so hoch wie die mit dem
    Element
```

```
    // korrespondierende Temperatur.
    if (adc_value >= conversion_table[celsius]))
    {
      // Da die Tabelle mit 0 beginnt, die Werte der Elemente jedoch
    mit -20,
      // muss der Wert geshiftet werden.
      return celsius + min_temp;
    }
  }

  // Wurde der Wert nicht gefunden, wird die minimale Temperaur
    ausgegeben.
  return min_temp;
}
```

Der Algorithmus sucht den Bereich aus der Tabelle in dem der ACD Wert liegt und wählt die niedrigere Nummer dieses Bereiches. Die Ranknummer markiert die Gradzahl und durch das Addieren der anfänglichen Temperatur wird eine Genauigkeit von 1°C erreicht.

Umrechnungstabelle und Funktion sind schon in der HomeLab Bibliothek enthalten, sie müssen somit nicht extra für diese Aufgabe erstellt werden. In der Bibliothek heißt die Umrechnungsfunktion *thermistor_calculate_celsius*. Hierbei muss beachtet werden, dass die Funktion nur korrekt ist, wenn sie mit dem Thermistor des Sensormoduls aus dem HomeLab verwendet wird. Werden andere Thermistoren genutzt, muss eine entsprechende Umrechnungstabelle erstellt und eine komplexere Funktion verwendet werden, welche im Handbuch der Bibliothek beschrieben wird. Das Beispielprogramm dieser Übung ist ein Thermometer, welches Temperatur in °C mißt und sie auf dem alphabetischen LCD ausgibt.

```
//
// Beispielprogramm des Thermistors des Sensormoduls.
// Die Temperatur wird auf dem LCD angezeigt.
//
#include <stdio.h>
#include <homelab/adc.h>
#include <homelab/module/sensors.h>
#include <homelab/module/lcd_alpha.h>

//
// Hauptprogramm
//
int main(void)
{
  unsigned short value;
  signed short temperature;
  char text[16];

  // Einrichten des LCD
  lcd_alpha_init(LCD_ALPHA_DISP_ON);
```

```
  // Löschen des LCD
  lcd_alpha_clear();

  // Name des Programms
  lcd_alpha_write_string("Termomeeter");

  // Einrichten des ADC
  adc_init(ADC_REF_AVCC, ADC_PRESCALE_8);

  // Endlosschleife
  while (true)
  {
    // Auslesen der 4-fach gerundeten Spannungswerte aus dem Thermistor
    value = adc_get_average_value(2, 4);

    // Konvertieren der ADC Werte in Grad Celsius
    temperature = thermistor_calculate_celsius(value);

    // Konvertieren der Temperatur in Text.
    // To display the degree sign, the octal variable is 337.
    sprintf(text, "%d\337C   ", temperature);

    // Anzeige des Textes am Anfang der zweiten Zeile des LCDs.
    lcd_alpha_goto_xy(0, 1);
    lcd_alpha_write_string(text);
  }
}
```

Extra

Das Temperaturdiagramm eines 10 kΩ NTC Thermistors

6.5.3 Fotowiderstand

Notwendiges Wissen: *Sensors Module,* *lcd,* *Voltage Divider,* *Analog-to-digital Converter,* *Analog to Digital Converter,* *Alphanumeric LCD,* *windows*

Theorie

Abbildung 6.18: Ein Fotowiderstand

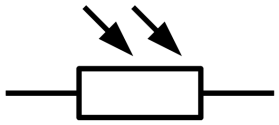


Abbildung 6.19: Elektrisches Symbol für einen Fotowiderstand

Ein Fotowiderstand ist ein Sensor, dessen elektrischer Widerstand sich je nach einfallender Lichtintensität verändert. Je intensiver das Licht ist, desto mehr freie Ladungsträger werden gebildet, und umso geringer wird dadurch der Widerstand des Bauteils. Durch das keramische Basismaterial führen zwei externe Metallkontakte des Widerstands zur lichtempfindlichen Membran, dessen Widerstand abhängig von der Geometrie und den Materialeigenschaften ist. Da lichtempfindliches Material durch die schmale, kurvige Spur zwischen den Elektroden bereits einen hohen Widerstand hat, kann der niedrigste totale Widerstand schon bei durchschnittlichen Lichtintensitäten erreicht werden. Der Fotowiderstand reagiert ähnlich dem menschlichen Auge nur auf bestimmte Wellenlängen, was bei der Auswahl eines solchen Bauteils bedacht werden muss. Andernfalls kann es vorkommen, dass der Widerstand nicht auf die in der Anwendung genutzte Lichtquelle reagiert. Nachfolgend ist eine Tabelle abgebildet, welche vereinfacht die Wellenlängen des sichtbaren Lichts mit den entsprechenden Farben darstellt:

Farbe	**Wellenlängen (nm)**
Violett	400 - 450
Blau	450 - 500
Grün	500 - 570
Gelb	570 - 590
Orange	590 - 610
Rot	610 - 700

Der Fotowiderstand arbeitet innerhalb eines bestimmten, festgelegten Temperaturbereichs. Soll der Sensor bei anderen Temperaturen genutzt werden, müssen präzise Umrechnungen durchgeführt werden, da die Widerstandseigenschaften des Sensors abhängig von der Umgebungstemperatur sind.

Zur Kennzeichnung der Lichtintensität wird die Beleuchtungsstärke (E) genutzt. Diese zeigt die Menge Licht an, die auf eine bestimmte Oberfläche trifft. Die Maßeinheit ist Lux (lx), wobei 1 Lux dem konstanten Lichtfluss von einem 1 Lumen entspricht, welcher auf eine Oberfläche von $1m^2$ strahlt. In der Realität fällt Licht jedoch

eigentlich nie gleichmäßig auf eine Oberfläche weshalb die Beleuchtungsstärke meistens als Durchschnittswert ermittelt wird. Unten sind ein paar Beispiele von Beleuchtungsstärken dargestellt:

Vergleichswerte von Beleuchtungsstärken:

Umgebung	**Beleuchtungsstärke (lx)**
Vollmond	0,1
Abenddämmerung	1
Auditorium	10
Klassenraum	30
Sonnenaufgang / -untergang	400
OP-Saal (Krankenhaus)	500 - 1000
direktes Sonnenlicht	10000

Übung

Das Sensormodul aus dem HomeLab verfügt über einen VT935G Fotowiderstand. Ein Pin des Widerstands ist an der +5 V Stromversorgung angeschlossen, der zweite an Kanal 1 (Pin PF1) des ADC. Zwischen diesem Pin und der Masse ist ein 10 kΩ Widerstand angeschlossen, welcher zusammen mit dem Fotowiderstand einen Spannungsteiler erzeugt. Da sich der elektrische Widerstand des Fotowiderstands mit einfallender Lichtintensität verkleinert, wird die gemessene Spannung am Pin des Mikrocontrollers mit dem Ansteigen der Lichtstärke größer. Hierbei sollte bedacht werden, dass der Fotowiderstand des HomeLab am stärksten auf oranges und gelbes Licht reagiert.

Der Sensor VT935G ist nicht als spezifisches Messinstrument gedacht. Er soll vielmehr dazu dienen, Informationen über die allgemeinen Lichtverhältnisse (z. B. befindet sich eine eingeschaltete Lampe im Raum oder nicht) anzugeben. Dazu muss nur der Widerstand des Sensors in einem halbdunklen Raum gemessen und ins Programm eingetragen werden. Daraufhin kann man die gemessenen Werte vergleichen und feststellen ob es heller oder dunkler ist.

Die nun folgende Aufgabe ist ein wenig komplexer, da die Beleuchtungsstärke auch in Lux gemessen wird. Hierzu werden eine Annäherungsformel sowie Gleitkomma-Variablen angewendet. In C sind Gleitkomma-Variablen *float*- und *double*--Typ Variablen, welche zur Darstellung von Brüchen genutzt werden können. Nachteilig ist jedoch, dass während der Nutzung dieser Variablen viele Ressourcen in Anspruch genommen werden müssen. Während Computer spezielle Hardware zur Berechnung von Gleitkomma-Variablen besitzen, wird diese bei den 8-Bit AVR Microcontrollern mittels einer Software durchgeführt, wodurch viel Zeit und Speicher benötigt wird. Sofern die Nachteile jedoch nicht entscheidend sind, lohnt es sich, Gleitkomma-Variablen zu verwenden.

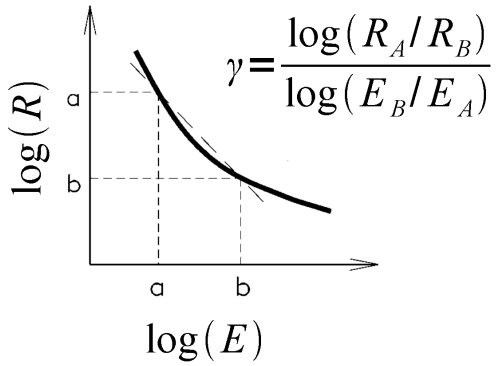


Abbildung 6.20: Verhältnis zwischen Widerstand (R) des VT935G und Lichtintensität (E)

Das Datenblatt des Sensors beinhaltet eine Annäherungsformel für den Zusammenhang zwischen Beleuchtungsstärke und elektrischem Widerstand. Wie im logarithmisch skalierten Graphen rechts dargestellt, sind Widerstand und Beleuchtungsstärke nahezu linear abhängig und bilden eine lineare Funktion, wenn folgende Konversion angewendet wird:

log(a/b) = log(a) - log(b)

Die Relation wird durch den Anstieg des Y-Faktors (Steigung der Geraden) beschrieben, welcher für den VT935G Sensor den Wert 0,9 hat. Darüber hinaus ist ein Punkt auf dieser Geraden bekannt: Der Widerstand beträgt 18.5 kΩ (R_A) bei 10 lx Beleuchtungsstärke (E_A). Mittels dieser Koordinaten und der Steigung der Geraden lässt sich nun jeder beliebige Punkt berechnen. Das bedeutet, wird der Widerstand (R_B) am Sensor gemessen, ist es möglich die Beleuchtungsstärke E_B) wie folgt mit der Formel der Geraden zu berechnen.

$\log(E_B) = \log(R_A/R_B) / \gamma + \log(E_A)$

$E_B = 10^{log(R_A/R_B)/\gamma + log(E_A)}$

So erhält man die Formel zur Berechnung der Beleuchtungsstärke bei gegebenem Widerstand. Da der Widerstand nicht direkt von Microcontroller gemessen werden kann befindet sich der Fotowiderstand im Spannungsteiler. Die Output-Spannung des Spannungsteilers wird durch den ADC zu einer spezifischen Variable konvertiert. Zur Bestimmung des Widerstands muss diese Output-Spannung (U_2) zunächst mittels des ADC-Wertes berechnet werden. Dabei muss auch die Vergleichs-Spannung des Konverters berücksichtigt werden. Folgende Formel wird zur Berechnung verwendet:

$U_2 = U_{ref} * (ADC / 1024)$

Mit Hilfe der Formel für Spannungsteiler (siehe Kapitel über Spannungsteiler) kann der Widerstand des Fotowiderstands (R_1) ermittelt werden:

$R_1 = (R_2 * U_1) / U_2 - R_2$

In der folgenden Berechnung von Spannung und Widerstand, werden die gegebenen Werte eingesetzt und die Indizes entfernt:

U = 5 * (ADC / 1024)

R = (10 * 5) / U - 10

Die Beleuchtungsstärke kann dann mittels der folgenden vereinfachten Konversion genutzt werden:

$E = 10^{log(18.5/R)/0.9+1} = 10^{log(18.5/R)*10/9} * 10^1 =$

$= 10^{log18.5*10/9-logR*10/9} * 10 = (10^{log18.5*10/9} / 10^{logR*10/9}) * 10 =$

$= (18.5^{10/9} / R^{10/9}) * 10 = 18.5^{10/9} * 10 * R^{-10/9}$

Durch die Berechnung der Konstante vor der Variable des Feldes R, bleibt folgender Ausdruck:

$E = 255{,}84 * R^{-10/9}$

Die dargestellten Formeln sind jedoch nur im Zusammenhang mit Fotowiderständen des HomeLab Sensor-Moduls nützlich. Wird ein Schaltkreis mit anderen Komponenten bestückt, müssen die Variablen verändert werden.Nachfolgend ist der Quellcode eines Beispielprogramms dargestellt, welches die Beleuchtungstärke mit Hilfe des ADC misst und berechnet und auf dem LCD darstellt. Bevor das Programm kompiliert wird, müssen die Einstellungen für die Gleitkomma-Variablen im Projekt festgelegt werden. Dieser Schritt wird in dem Kapitel über die Installation der Software erläutert.

In diesem Beispielprogramm werden die Variablen für Spannung, Widerstand und Beleuchtungsstärke als *double* Gleitkomma-Variablen definiert. Die Variablen, welche nun als Gleitkomma-Variablen genutzt werden, müssen immer eine Dezimalstelle beinhalten (diese kann auch einfach 0 sein, so verarbeitet der Compiler es korrekt). Wird die Funktion *sprintf* genutzt, um die Gleitkomma-Variable in Text zu konvertieren, muss das "%f"Format genutzt werden, welches durch Nutzung von ganzen Zahlen und Dezimalstellen erweitert werden kann. So gibt "%3.2"beispielsweise immer 3 ganze Zahlen und 2 Dezimalstellen an.

```
//
// Beispielprogramm des Fotowiderstands aus dem Sensormodul des HomeLab
// Der angenäherte Wert der Beleuchtungsstärke wird auf dem LCD
    dargestellt.
//
#include <stdio.h>
#include <math.h>
#include <homelab/module/lcd_alpha.h>
#include <homelab/adc.h>
#include <homelab/delay.h>

//
// Hauptprogramm.
//
int main(void)
{
  char text[16];
```

```
  unsigned short adc_value;
  double voltage, resistance, illuminance;

  // Initialisiere LCD
  lcd_alpha_init(LCD_ALPHA_DISP_ON);

  // Lösche LCD.
  lcd_alpha_clear();

  // Name des Programms
  lcd_alpha_write_string("Luxmeter");

  // Einrichten des ADC
  adc_init(ADC_REF_AVCC, ADC_PRESCALE_8);

  // Endlosschleife.
  while (true)
  {
    // Durchschnittlichen Wert des Fotoresistors auslesen
    adc_value = adc_get_average_value(1, 10);

    // Input-Spannung am ADC berechnen
    voltage = 5.0 * ((double)adc_value / 1024.0);

    // Widerstand des Fotoresistors im Spannungsteiler berechnen
    resistance = (10.0 * 5.0) / voltage - 10.0;

    // Beleuchtungsstärke in lux berechnen
    illuminance = 255.84 * pow(resistance, -10/9);

    // Beleuchtungsstärke in Text konvertieren
    sprintf(text, "%0.1f lux   ", illuminance);

    // Darstellung auf dem LCD
    lcd_alpha_goto_xy(0, 1);
    lcd_alpha_write_string(text);

    // Verzögerung 500 ms
    sw_delay_ms(500);
  }
}
```

6.5.4 Infrarot-Entfernungsmesser

Notwendiges Wissen: *Sensors Module,* *lcd,* *Analog-to-digital Converter,* *Analog to Digital Converter,* *Alphanumeric LCD,* *Sensors*

Theorie

Abbildung 6.21: Sharp GP2Y0A21YK

Um die Distanz zu einem Objekt zu messen, gibt es optische Sensoren, welche die Triangulation als Messmethode nutzen. Die am häufigsten verwendeten Infrarot-Entfernungsmesser werden von der Firma „Sharp“produziert und verfügen über einen analogen Spannungs-Output. Diese Sensoren besitzen eine Infrarot-LED mit einer Linse, welchen einen schmalen Lichtstrahl aussendet. Dieser wird von dem Objekt, zu welchem die Distanz gemessen werden soll, reflektiert und dann durch die zweite Linse zu einem optischen Positionssensor (OPS) gelenkt. Die Leitfähigkeit dieses OPS ist abhängig davon, an welcher Stelle der Lichtstrahl einfällt. Sie wird in Spannung umgewandelt, die Spannung wird mittels eines Analog-Digital-Konverters digitalisiert und daraufhin kann die Entfernung berechnet werden. Die Zeichnung auf der rechten Seite zeigt die Wege von Strahlen, die aus unterschiedlichen Entfernungen reflektiert werden.

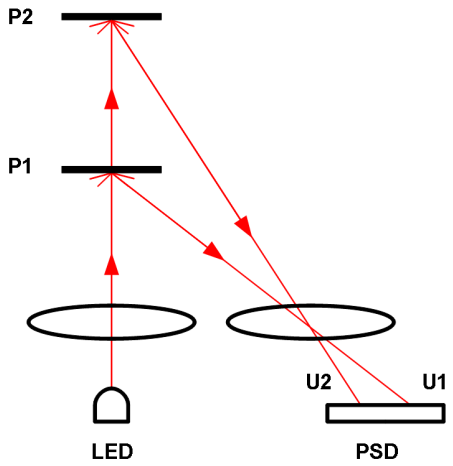


Abbildung 6.22: Wege der Lichtstrahlen eines Infrarot-Entfernungsmessers

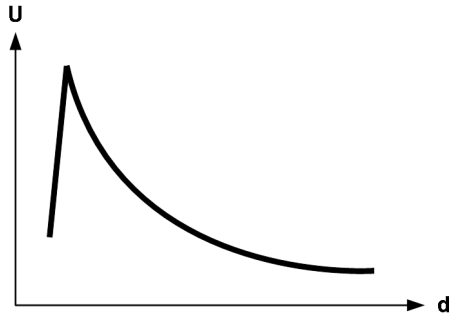


Abbildung 6.23: Das Verhältnis von Spannung und Entfernung eines Infrarot-Entfernungsmessers

Der Output der Infrarot-Entfernungsmesser von „Sharp“ist anti-proportional. Das meint, mit größer werdender Entfernung verringert sich der Output (die Abnahme des Outputs, also der Spannung, wird jedoch nach und nach schwächer). Der Graph mit dem exakten Verhältnis zwischen Entfernung und Spannung ist gewöhnlich im Datenblatt des Sensors abgebildet.Jeder Sensor-Typ hat seinen spezifischen Messbereich, innerhalb welchem die gemessenen Ergebnisse verlässlich sind. Die maximale Messdistanz wird von zwei Aspekten beschränkt: Zum Einen von der Abnahme der Menge des reflektierten Lichts, zum Anderen vom Unvermögen des OPS, kleine Positonsänderungen des reflektierten Strahls zu registrieren. Misst man die Entfernung zu Objekten, welche zu weit entfernt sind, entspricht der Output nahezu dem, bei der Messung zur maximalen Distanz.Die minimale Entfernung wird durch die Besonderheit der Sharp Sensoren eingeschränkt. So fällt der Output steil ab, sobald die Entfernung einen bestimmten Punkt unterschreitet (je nach Baureihe liegt dieser bei 4 bis 20 cm) Das hat zur Folge, dass zu einem Outputwert zwei korrespondierende Entfernungswerte existieren.Dieses Problem kann vermieden werden, indem beachtet wird, dass die Distanz zwischen Objekt und Sensor nicht zu gering ist.

Übung

Das HomeLab Sensor-Kit enthält den Infrarot-Entfernungsmesser SHARP GP2Y0A21YK. Der Messbereich dieses Sensors liegt zwischen 10 cm und 80 cm. Seine Output-Spannung hängt von der Entfernung ab und erreicht bis zu 3 V. Der Entfernungssensor ist am Sensormodul angeschlossen. Die Output-Spannung wird über Kanal 0 zum ADC des AVR gesendet. Aufbauend auf den vorangehenden Übungen zu Sensoren, ist es einfach ein Programm zu schreiben, welches die Output-Spannung des Entfernungsmessers misst. Zusätzlich dazu behandelt diese Aufgabe die Umwandlung der Spannung in eine Entfernung.

Das Datenblatt des GP2Y0A21YK beinhaltet einen Graphen welcher die Relation von Output-Spannung und gemessener Entfernung darstellt. Dieser Graph verläuft nicht linear. Der Graph der inversen Werte ist jedoch annähernd linear, wodurch relativ einfach die Formel zur Umwandlung von Spannung in Entfernung gefunden werden kann. Hierzu werden Punkte des Graphens in ein Tabellenkalkulationsprogramm eingefügt und so ein neuer Graph generiert. Die meisten Programme berechnen automatisch eine Trendlinie.Nachfolgend ist der Graph des GP2Y0A21YK mit der Relation der korrigierten inversen Werte der Outputspannung zu den korrigierten inversen Werten der gemessenen Distanz, inklusive einer Trendlinie abgebildet. Zur Vereinfachung wurde die Output-Spannung schon in 10-Bit +5 V Werte des ADC mit Vergleichspannung konvertiert.

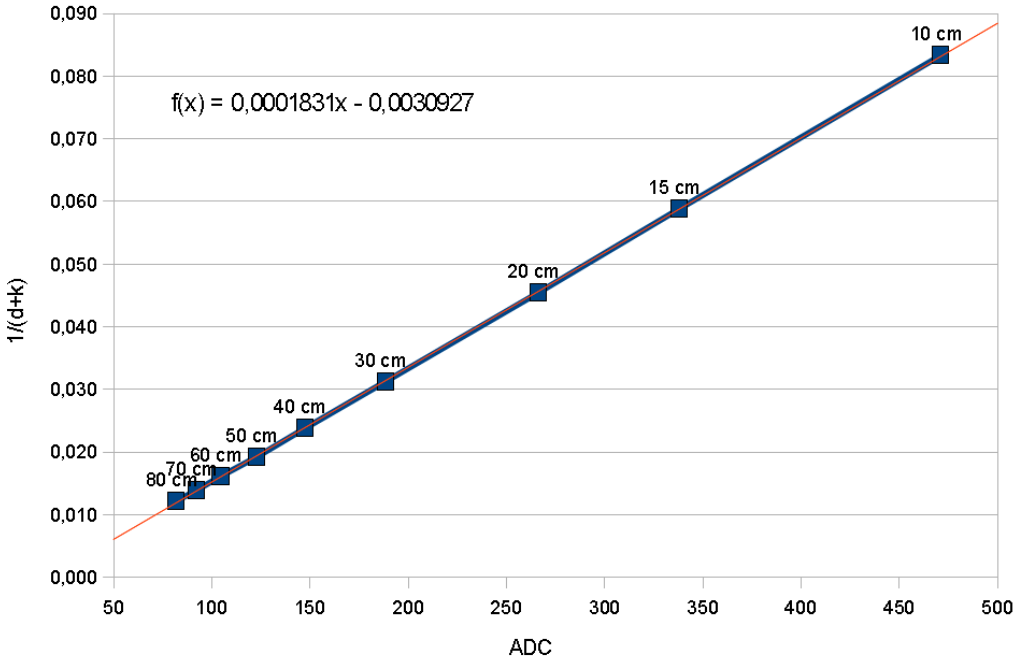


Abbildung 6.24: Graph zur linearen Darstellung von ADC Wert und Entfernung

Wie der Graph zeigt, überschneiden sich die blaue Trendlinie und die Punkte des Graphen fast genau. Dieses wird durch die Nutzung einer Korrektur-Konstante erreicht. Diese Konstante wird mit der „trail-and-error"-Methode gefunden - es werden dabei viele Variablen getestet, bis diejenige gefunden ist, die den Graph mit der Trendlinie überlappen lässt.Im abgebildeten Graph ist die Korrektur-Konstantet +2; das bedeutet, zu allen realen Entfernungen muss +2 hinzuaddiert werden. Daruch verläuft der Graph nahezu gleich der Trendline und

es kann für die Relation zwischen Entfernung und Spannung verallgemeinert folgende Formel angenommen werden:

1 / (d + k) = a * ADC + b

wobei

d - Entfernung in cm.

k - Korrektur-Konstante (gefunden mittels trial-and-error Methode)

ADC - digitalisierter Wert der Spannung.

a - lineares Element (Wert wird durch die Trendlinien-Gleichung bestimmt)

b - freies Element (Wert wird durch die Trendlinien-Gleichung bestimmt)

Die Entfernung d kann durch folgende Formel dargestellt werden:

d = (1 / (a * ADC + B)) - k

Nun ist es generell möglich die Entfernung mit der Formel zu berechnen. Da jedoch Brüche dividiert werden, sind zudem Gleitkomma-Berechnungen nötig. Weiterhin muss die Formel vereinfacht und auf größere Quotienten ausgeweitet werden, da Microcontroller mit Ganzzahlen arbeiten. Durch die Division des Quotienten mit einem linearen Element erhält man folgende Formel:

d = (1 / a) / (ADC + B / a) - k

Durch Aufnahme der Korrekturkonstante sowie des linearen und freien Elements der Trendlinien-Gleichung, ergibt sich folgende Formel zur Berechnung der Entfernung:

d = 5461 / (ADC - 17) - 2

Diese Formel kann mit 16-Bit Zahlen berechnet werden und ist daher vollständig für den AVR geeignet. Vor der Berechnung muss sicher gestellt sein, dass der Wert des ADC größer als 17 ist, ansonsten kann der Fall eintreten, dass durch 0 geteilt werden muss oder eine negative Distanz ausgegeben wird.

Nachfolgend ist die Funktion dargestellt, die dazu dient, ADC-Werte in Centimeter zu konvertieren. Sie ist in der HomeLab Library enthalten. Lineare und freie Elemente sowie die Korrektur-Konstante, sind nicht fest in die Funktion integriert, sondern werden mittels der Objektparameter des Infrarot-Entfernungsmessers eingespeist. Dadurch, dass diese Parameter als separate Konstanten verwendet werden, ist es sehr einfach, neue Infrarot-Entfernungsmesser in das Programm zu integrieren.

```
//
// Die Struktur der Parameter des Infrarot-Entfernungsmessers
//
typedef const struct
{
  const signed short a;
```

```
  const signed short b;
  const signed short k;
}
ir_distance_sensor;

//
// Die Parametereigenschaften des GP2Y0A21YK Sensors
//
const ir_distance_sensor GP2Y0A21YK = { 5461, -17, 2 };

//
// Konvertieren der Werte des Infrarot-Entfernungsmessers in Centimeter
// Gibt -1 aus, wenn die Konvertierung nicht erfolgreich war
//
signed short ir_distance_calculate_cm(ir_distance_sensor sensor,
  unsigned short adc_value)
{
  if (adc_value + sensor.b <= 0)
  {
    return -1;
  }

  return sensor.a / (adc_value + sensor.b) - sensor.k;
}
```

Zur Durchführung der Konvertierung muss die Funktion *ir_distance_calculate_cm* genutzt werden. Der erste Funktionsparameter gibt die Eigenschaften der Parameter des Infrarot-Entfernungsmessers an, der zweite den ADC-Wert. Die Funktion gibt dann die berechnete Entfernung in Centimetern aus. Schlägt die Konvertierung fehl (unzulässige ADC-Werte), wird der Wert -1 ausgegeben. Das folgende Programm veranschaulicht die Verwendung des Infrarot-Entfernungsmessers und der Konvertierungsfunktion. Es wird das alphanumerische LCD genutzt, auf welchem die gemessenen Werte dargestellt werden. Falls die Distanz unzulässig ist, wird ein "?"angezeigt.

```
//
// Beispielprogramm des Infrarot-Entfernungsmessers des HomeLab
// Die in Centimetern gemessenen Ergebnisse werden auf dem LCD
    abgebildet
//
#include <stdio.h>
#include <homelab/adc.h>
#include <homelab/delay.h>
#include <homelab/module/sensors.h>
#include <homelab/module/lcd_alpha.h>

//
// Hauptprogramm
```

```
//
int main(void)
{
  unsigned short value;
  signed short distance;
  char text[16];

  // External sensor selection
  pin ex_sensors = PIN(G, 0);
  pin_setup_output(ex_sensors);
  pin_set(ex_sensors);

  // Initialisierung des LCD
  lcd_alpha_init(LCD_ALPHA_DISP_ON);

  // Löschen des LCD
  lcd_alpha_clear();

  // Name des Programms
  lcd_alpha_write_string("Distance sensor");

  // Installation des ADC
  adc_init(ADC_REF_AVCC, ADC_PRESCALE_8);

  // Endlosschleife
  while (true)
  {
    // Auslesen des viermal gerundeten Wertes der Outputspannung des
    Sensors
    value = adc_get_average_value(0, 4);

    // Konvertieren des ADC-Wertes in Entfernung
    distance = ir_distance_calculate_cm(GP2Y0A21YK, value);

    // War die Berechnung erfolgreich?
    if (distance >= 0)
    {
      // Konvertieren von Entfernung in Text
      sprintf(text, "%d cm    ", distance);
    }
    else
    {
      // Text für eine unbekannte Entfernung erzeugen
      sprintf(text, "? cm    ");
    }
```

```
    // Anzeige des Textes am Anfang der zweiten Zeile auf dem LCD
    lcd_alpha_goto_xy(0, 1);
    lcd_alpha_write_string(text);

    // Break
    sw_delay_ms(500);
  }
}
```

Zusätzliches Material

Graph des Sharp GP2Y0A21YK Sensors

Linearisierung der Sharp Entfernungsdaten

6.5.5 Ultraschall-Entfernungsmesser

Notwendiges Wissen: *Controller module,* *lcd,* *Counters/Timers,* *Timers,* *Alphanumeric LCD,* *Sensors*

Theorie

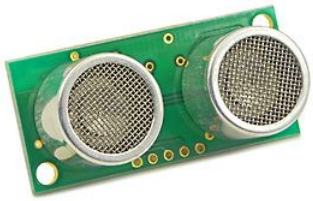


Abbildung 6.25: Ultraschall-Entfernungsmesser SRF04

Ein Ultraschall-Entfernungsmesser bestimmt die Entfernung zu einem Objekt, indem er die Zeit misst, die ein Schall zu einem Objekt hin und zurück benötigt. Die Frequenz des Geräusches liegt im Bereich des Ultraschalls. Dies garantiert, dass die Schallwelle gebündelt gelenkt wird, da sich hochfrequenter Schall weniger in der Umgebnung zerstreut. Ein typischer Ultraschall-Entfernungsmesser besteht aus zwei Membranen. Eine Membran produziert den Schall, die andere empfängt das Echo. Im Grunde besteht er also aus einem Lautsprecher und einem Mikrophon. Der Schallgenerator generiert kurze (wenige Perioden umfassende) Ultraschallimpulse und

startet den Timer. Die zweite Membran registriert die Ankunft des Echos und stoppt den Timer. Mit der Zeit des Timers ist es möglich, die zurückgelegte Entfernung des Schalls zu berechnen. Die Entfernung zu einem Objekt entspricht der Hälfte der zurückgelegten Entfernung des Schalls.

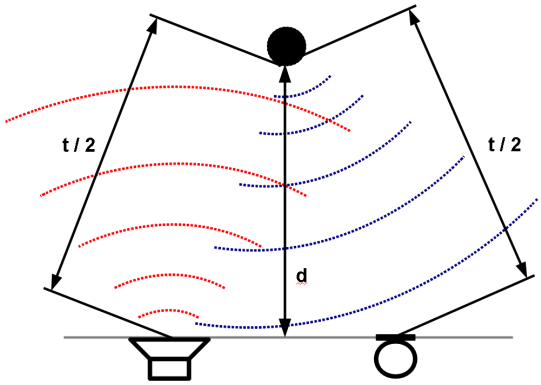


Abbildung 6.26: Funktionsweise eines Ultraschall-Entfernungsmessers.

Der Ultraschall-Entfernungsmesser findet im täglichen Leben eine Vielzahl von Anwendungsmöglichkeiten. Er wird beispielsweise als Ersatz für Maßbänder auf Baustellen eingesetzt, Autos besitzen Ultraschall-Entfernungsmesser als Parksensoren. Neben der Entfernungsmessung dient er auch dazu, das bloße Vorhandensein eines Objekts im Messbereich zu erkennen, z. B. in der Gefahrenzone einer Arbeitsmaschine. Sind Ultraschalltransmitter und -empfänger getrennt, kann die Fließgeschwindigkeit des Materials zwischen ihnen bestimmt werden, da Schallwellen langsamer aufwärts fließen.

Übung

Das HomeLab enhält einen Devantech SRF04/SRF05 Ultraschall-Entfernungsmesser. Der SRF04/SRF05 ist jedoch nur ein Sensor und gibt keine direkte Information über die Distanz. Neben den Stromversorgungspins besitzt der Sensor auch einen Trigger- und einen Echo-Pin.Ist der Trigger-Pin „high“, generiert der Sensor eine acht Perioden lange 40 kHz Ultraschallwelle. Daraufhin wird der Echo-Pin „high“und bleibt „high“, bis das Echo zurück kommt. Das Echo-Signal gibt also generell die Zeit an, die der Schall benötigt um das Objekt zu erreichen und zum Sensor zurück zu kommen. Die Entfernung zum Objekt ergibt sich daraus, dass die gemessene Zeit mit der Schallgeschwindigkeit multipliziert und dann durch zwei dividiert wird. Der folgende Graph zeigt den Zusammenhang zwischen Zeit und den Signalen des Empfängers, Triggers und des Echos:

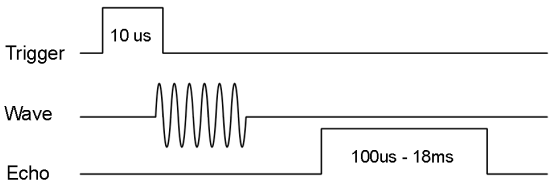


Abbildung 6.27: Die Signale des SRF04

Um den SRF04/SRF05 mit dem AVR zu nutzen, müssen Trigger- und Echo-Pin an den AVR Pins angeschlossen werden. Zur Zeitmessung, sollte ein 16-Bit Timer genutzt werden, z.B. *timer3*. Nachfolgend ist eine Funktion dargestellt, die alle Messvorgänge ausführt - sie generiert das Signal des Triggers, startet den Timer, misst die Länge des Echosignals und konvertiert dies zur Entfernung in Centimetern. Die Funktion blockt, das bedeutet sie hält den Prozessor beschäftigt, bis die Messung abgeschlossen ist oder zu lange dauert. Je schneller das Echo ankommt, desto schneller erhält man ein Ergebnis. Falls kein Echo ankommt, wartet die Funktion 36 ms und geht dann auf 0 zurück. Es ist wichtig zwischen den Messungen eine Pause von mindestens 20 ms Pause einzuhalten, um den Soundgenerator komplett zum Stillstand kommen zulassen, so dass die neue Messung nicht durch die alte beeinträchtigt wird.Außerdem sollte darauf geachtet werden, dass die Schallwellen sich nicht gegenseitig stören, wenn verschiedene Ultraschallsensoren gleichzeitig genutzt werden.

```
#define ULTRASONIC_SPEED_OF_SOUND 33000 // cm/s

//
// Sofortige Ultraschall-Entfernungsmessung
//
unsigned short ultrasonic_instant_measure(pin trigger, pin echo)
{
  // Pin-Setup
  pin_setup_output(trigger);
  pin_setup_input_with_pullup(echo);

  // Timer 3 auf den normalen Modus setzen
  // mit der Periode des F_CPU / 8
  timer3_init_normal(TIMER3_PRESCALE_8);

  // Erzeugung des Aulöse-Impulses
  pin_set(trigger);

  // Timer zurücksetzen
  timer3_overflow_flag_clear();
  timer3_set_value(0);

  // ~10 us warte
  while (timer3_get_value() < 18) {}

  // Auslöse-Impuls beenden
  pin_clear(trigger);

  // Warten auf den Echo-Start
  while (!pin_get_value(echo))
  {
    // Timeout ?
    if (timer3_overflow_flag_is_set())
    {
      return 0;
    }
```

```
  }

  // Timer erneut zurücksetzen
  timer3_set_value(0);

  // Warten auf Beendigung des Echos
  while (pin_get_value(echo))
  {
    // Timeout ?
    if (timer3_overflow_flag_is_set())
    {
      return 0;
    }
  }

  // Konvertierung von Zeit in Entfernung:
  //   Entfernung = Timer * (1 / (F_CPU / 8)) * Geschwindigkeit / 2
  return (unsigned long)timer3_get_value() *
    ULTRASONIC_SPEED_OF_SOUND / (F_CPU / 4);
}
```

Die gegebene Funktion erlaubt dem Nutzer den Echo-und den Trigger-Pin so zu wählen, dass der Sensor dort angeschlossen werden kann wo es am günstigesten ist und genug Platz vorhanden ist. Zusätzlich erlaubt dies, die Funktion auch außerhalb von HomeLab zu nutzen. Die dargestellte Funktion ist bereits in der HomeLab Library enthalten und muss somit nicht geschrieben werden. Es gibt noch eine Sache zu beachten: Die Funktion in der Library ist streng an die Taktrate des Controller-Moduls von HomeLab gekoppelt. Die Taktrate beträgt 14,7456 MHz, wenn die Funktion mit anderen Taktraten genutzt wird, liefert sie falsche Ergebnisse. Um die Funktion mit anderen Taktraten zu nutzen sollte sie manuell programmiert werden. Der folgende Code demonstriert die Nutzung des SRF04/SRF05 Ultraschall-Entfernungsmessers mit der HomeLab Library:

```
//
// Beispielprogramm für den Ultraschall-Entfernungsmesser des HomeLab
// Die Entfernungsmessung wirkt blockierend.
//
#include <stdio.h>
#include <homelab/pin.h>
#include <homelab/delay.h>
#include <homelab/module/sensors.h>
#include <homelab/module/lcd_alpha.h>

//
// Pins des Ultraschallsensors
//
pin pin_trigger = PIN(G, 1);
pin pin_echo    = PIN(G, 0);

//
```

```
// Hauptprogramm
//
int main(void)
{
  unsigned short distance;
  char text[16];

  // Initialisierung des LCD
  lcd_alpha_init(LCD_ALPHA_DISP_ON);

  // Löschen des LCD
  lcd_alpha_clear();

  // Name des Programms
  lcd_alpha_write_string("Ultra sound");

  // Kleine Unterbrechung
  sw_delay_ms(100);

   // Endlosschleife.
  while (true)
  {
    // Messen
    distance = ultrasonic_measure(pin_trigger, pin_echo);

    // War die Messung erfolgreich?
    if (distance > 0)
    {
      // Konvertierung von Entfernung in Text.
      sprintf(text, "%d cm    ", distance);
    }
    // Sind während der Messung Fehler aufgetreten?
    else
    {
      // Text des Fehlers.
      sprintf(text, "Error    ");
    }

    // Darstellung des Textes am Anfang der zweiten Zeile des LCD
    lcd_alpha_goto_xy(0, 1);
    lcd_alpha_write_string(text);

    // Kleine Unterbrechung.
    sw_delay_ms(500);
  }
```

```
}
```

6.5.6 Übungen

Ziel ist es, ein ein Programm zu schreiben welches folgende Aufgaben ausführt:

Aufwärmübung

Der Widerstand des Potentiometers soll auf dem LCD in Ohm angezeigt werden. Der nominale Wert des Widerstands beträgt 5 kΩ.

=== Für Anfänger =

1. Der Widerstand des Potentiometers wird im Bereich von (0 Ω. . . 999 Ω) in Ohm dargestellt und im Bereich von (1000 Ω. . . 5000 Ω) in kOhm. Der nominale Wert des Widerstands beträgt 5 kΩ. Das Ergebnis wird mit korrekten Einheiten und Symbolen dargestellt.

2. Messung der Distanz zu einem Objekt. Die Distanz zu dem Objekt wird mit einem Infrarot-Entfernungsmesser gemessen wenn der Schalter S1 betätigt wird. Während der Messung blinkt eine gelbe LED. Ist das Objekt weiter als 50 cm entfernt, leuchtet eine grüne LED, ist es weniger weit entfernt, die rote.

3. Die Distanz zu dem Objekt wird mit einem Infrarot-Entfernungsmesser gemessen. Das Ergebnis wird im binären System mit den 3 LEDs dargestellt (LED1, LED2, und LED3). Vergrößert sich die Entfernung, muss auch der angezeigte Wert größer werden. Als Skala sollte etwa 1 dm = 1 Bit verwendet werden.

4. Der Wert des NTC Temperatursensors wird auf dem LCD in Grad angezeigt. Durch Betätigung des Schalters S2 können die Einheiten ausgewählt werden: Kelvin (K), Fahrenheit (F) und Celsius (C). Die Temperatur wird in korrekten Einheiten und Symbolen angezeigt.

5. Mit dem Lichtintensitätssensor werden schnelle Änderungen der Lichtintensität festgestellt. (Licht an- / ausschalten). Verläuft die Änderung schnell, blinkt die rote LED für 5 Sekunden. Ändert sich die Intensität langsam, wird die Änderungsrichtung gezeigt. Die grüne LED zeigt an, dass die Lichtintensität ansteigt, die gelbe, dass sie abnimmt.

Für Fortgeschrittene

1. Datenschreiber. Die Werte aller analogen Sensoren werden konstant gemessen und die minimalen und maximalen Werte werden festgehalten. Mit Betätigung von S1 kann der Nutzer die auf dem LCD angezeigten Information wählen. Angezeigt werden muss: Der Name des Sensors(kurz) und die bislang erreichten minimalen / maximalen Werte. Die Sequenz des Durchschaltens ist folgende: IR Entfernungsmesser → Fotowiderstand → Thermistor → Potentiometer.

2. Entfernungsmesser. Wenn der Schalter S2 gedrückt wird, werden der Reihe nach 10 Messungen in einer Sekunde durchgeführt. Nach der Messung wird die durchschnittliche Entfernung zum Objekt in Dezimeter auf dem 7-Segment Indikator angezeigt. Bei Betätigung des Schalters S1, wird die minimale gemessene Entfernung angezeigt und durch S3 die maximale.

3. Geschwindigkeit. Die Änderungsrate der Entfernung zu einem Objekt wird wie folgt angezeigt: langsame Änderung - grüne LED, mittlere Änderung - gelbe LED, schnelle Änderung - rote LED. Die Geschwindigkeit wird auf dem LCD dargestellt.

4. Automatische Messreichweite. Je nach aktueller Messung werden die minimal oder maximal gemessenen Temperaturen augegeben und der zugehörige Wertebereich wird auf einer Skala von 0 bis 9 angezeigt. Der Wert wird auf der 7-Segment-Anzeige dargestellt.

5. Durch Kombination von Infrarot- und Ultraschall-Entfernungsmesser kann die Entfernung zum Objekt, die Geschwindigkeit sowie die Richtung des Objektes bestimmt werden. Die Ergebnisse werden auf dem LCD dargestellt.

Fragen

1. Wie exakt ist der ADC des ATmega128 Mikrocontrollers? Welches ist die kleinste noch messbare Veränderung der Inputspannung?

2. Wie lange dauert ein ADC-Prozess? Wie kann die Arbeitsfrequenz geändert werden?

3. Welchen Inputspannungsbereich hat der ADC? Kann dieser geändert werden? Wie?

4. Was ist der Unterschied zwischen PTC-Thermistoren und NTC-Thermistoren? Worin bestehen die jeweiligen Vorteile?

5. Was ist der Grund eines Spannungsteilers in einem Messschaltkreises?

6. Kombinieren Sie einen Spannungsteiler, welcher die Nutzung eines analogen Sensors mit dem ATmega128 Mikrocontrollers ermöglicht. Die maximale Outputspannung des Sensors beträgt 10 V. Ermitteln Sie zusätzlich den Inhalt des ADMUX Registers.

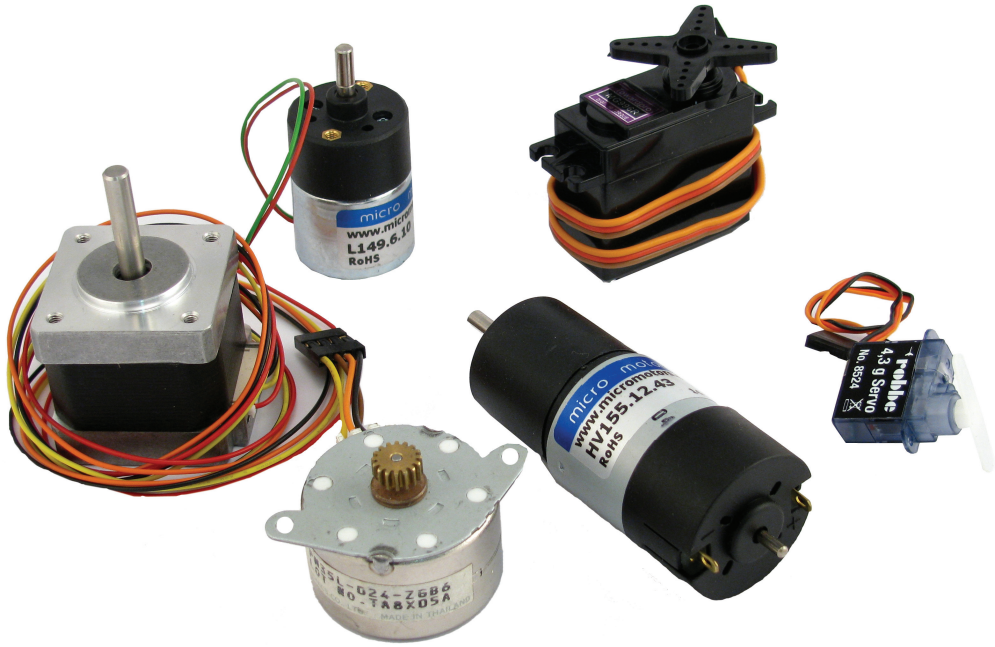
7. Zusätzliche Widerstände werden an den Pins des Potentiometers angeschlossen und eine 5 V Spannung wird angelegt. Wie groß müssen die zusätzlichen Widerstände sein, damit die Spannung vom Potentiometer zwischen 1 und 2 V reguliert werden kann (von einem Ende zum anderen)? Die Stromstärke darf 10 mA nicht überschreiten.

8. Welche Umgebungsparameter haben einen Effekt auf die Funktion des Ultraschallentfernungsmessers und warum?

9. Welche Lichtsensivitätssensoren können in einem Robotikprojekt genutzt werden? Nennen Sie mindestens 3 grundlegende Komponenten und erklären Sie die Unterschiede.

10. Wie kann, neben der trigonometrischen Methode, die Entfernung mittels Licht gemessen werden? Nennen Sie mindestens drei Methoden.

6.6 Motoren



Motoren sind Antriebsgeräte, sie können in jeder Form sehr verschieden sein, vom Arbeitsprinzip bis hin zu Kraft und Größe. In der Robotik werden hauptsächlich Elektromotoren benutzt. Diese wandeln elektrische Energie in mechanische Energie (Arbeit) um. Sie funktionieren auf dem Prinzip des Elektromagnetismus.

Es gibt viele Möglichkeiten, elektrische Motoren zu klassifizieren. Die bedeutendste Unterscheidung ist die zwischen AC- (Wechselstrom-) und DC- (Gleichstrom-) Motoren. Hinzu kommen Motoren mit Bürsten oder ohne, lineare Motoren und Rotationsmotoren, Nano-Motoren und große Motoren usw. Dahingegen erfolgen einige Klassifizierungen nur provisorisch. So werden z.B. lineare Bewegungen durch einen elektrischen Rotationsmotor bewerkstelligt, welcher mit Schrauben in einem einheitlichen Gehäuse verbaut ist und so als linearer Antrieb zählt. In diesem Kapitel werden die drei häufigsten Motoren, die in der Robotik zum Einsatz kommen,beschrieben: DC-Motoren mit Magneten, RC-Servomotoren und Schrittmotor.

6.6.1 Gleichstrommotor

Notwendiges Wissen: *Motor Module,* *Digital Inputs/Outputs,* *Motors,* *Delay*

Theorie

Abbildung 6.28: DC Motor

Ein Gleichstrommotor mit permanenten Magneten wird häufig für die verschiedensten Anwendungen genutzt, bei denen kleine Ausmaße, große Kraft und ein geringer Preis entscheidend sind. Auf Grund ihrer hohen Geschwindigkeit, werden sie oft mit einer Übersetzung genutzt. (Um eine niedrigere Output-Geschwindkeit und höheres Drehmoment zu erreichen.)

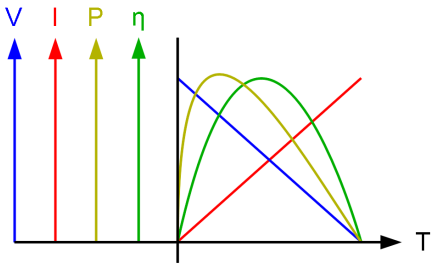


Abbildung 6.29: Der perfekte Graph des Verhältnisses von Geschwindigkeit (V), Stromstärke(I), Kraft(P), Effizienz (η) und Drehmoment (T) eines DC Motors.

Gleichstrommotoren mit permanenten Magneten sind vom Aufbau her sehr einfache und ihre Bedienung ist relativ einfach. Aber auch wenn die Kontrolle sehr einfach ist, wird die Geschwindigkeit nicht präzise über das Steuersignal bestimmt, weil hierauf noch viele andere Faktoren, wie das Drehmoment und der Speisestrom, einwirken. Das Verhältnis zwischen Drehmoment und Geschwindigkeit ist in einem idealen Gleichstrommotor linear. Das bedeutet, je höher das zu erzeugende Drehmoment ist, desto geringer die Geschwindigkeit und desto höher der Strom in der Spule.

Gleichstrommaschinen nutzen Gleichstrom und brauchen eigentlich keine spezielle Kontrollelektronik, da dienNotwendige Kommunikation im Motor selbst passiert. Wenn der Motor arbeitet gleiten zwei statische Schleifbürsten an dem sich drehenden Kommutator und halten die Spannung an den Spulen. Die Richtung in der sich der Motor dreht, hängt davon ab, wie der der Strom durch den Motor fließt. Wenn der Motor sich nur in eine Richtung bewegen muss kann der Strom über ein Relais oder ein einfache Schaltung kommen. Falls

der Motor sich in zwei Richtungen drehen muss, wird ein Schaltkreis, der Vierquadrantensteller (H-Bridge), genutzt.

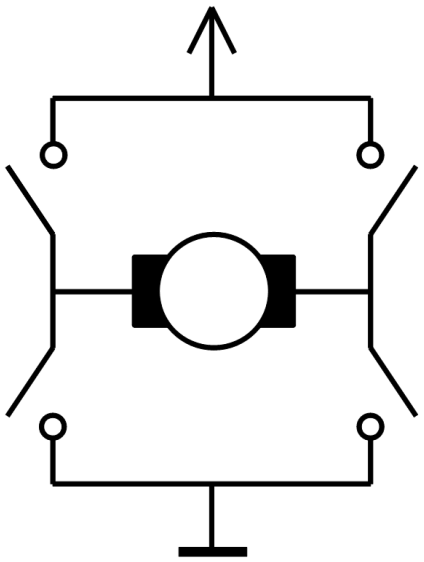


Abbildung 6.30: Die Arbeitsweise eines Vierquadrantenstellers, genutzt an Schaltern.

Der Vierquadrantensteller besitzt vier Transistoren die den Strom lenken, um den Motor zu betreiben. Das elektrische Schaltbild des Vierquadrantenstellers ist ähnlich des Buchstaben H, daher der englische Name H-Bridge. Das Besondere am Vierquadrantensteller ist, dass man damit beide Polaritäten am Motor anlegen kann. Das Bild auf der Seite zeit das grundsätzliche Schema eines Vierquadrantenstellers am Beispiel der Schalter. Werden zwei diagonale Schalter geschlossen, arbeitet der Motor. Die Drehrichtung des Motors hängt davon ab welche diagonalen Schalter geschlossen werden. Im wirklichen Vierquadrantensteller sind die Schalter durch Transistoren ersetzt, welche je nach Stromstärke und Spannung des Motors gewählt werden.

Zusätzlich zur Richtungsänderung kann der Vierquadrantensteller auch die Geschwindigkeit des Motors verändern. Dazu werden die Transistoren konstant durch Pulsweitenmodulation geöffnet und geschlossen. Damit ist die Energie, die der Motor erhält irgendwo zwischen vollem und keinem Stromdurchfluss. Die Zeit in der die Transitoren geöffnet sind, wird in der PWM-Periode Arbeitszyklus genannt, welche in % angegeben wird. Bei 0 % ist der Transistor konstant geschlossen und es fließt kein Strom. Bei 100 % ist der Transistor ist durchgehend geöffnet, und Strom fließt die ganze Zeit. Die Frequenz der PWM muss hoch genug sein, um Vibrationen der Motorwelle zu verhindern. Bei niedrigen Frequenzen produziert der Motor Geräusche, daher wird oft eine Modulationsfrequenz von über 20 kHz genutzt. Auf der anderen Seite ist die Effizienz des Vierquadrantenstellers nicht so gut bei hohen Frequenzen. Vibrationen der Motorwelle werden durch die Trägheit des Rotors und der Induktionswiderstand der Spulen reduziert.

Es gibt auch integrierte Vierquadrantensteller für niedrigere Stromstärken. Für höhere Stromstärken werden spezielle Metall-Oxid-Halbleiter-Feldeffekttransistoren verwendet. Der Vierquadrantensteller mit anderer Elektronik wird Motorcontroller oder Treiber genannt.Der Teiber des Gleichstrommotors im HomeLab L293D hat 2 eingebaute Vierquadrantensteller und Ausschaltdioden. Der Motor wird mit drei digitalen Signalen gesteuert, eins davon ist das *enable* Signal, welches Funktionen einschaltet, die anderen beiden bestimmten den

Status der Transistoren des Vierquadrantenstellers. Dabei dürfen niemals beide vertikalen Transistoren geöffnet werden, da dadurch ein Kurzschluss verursacht wird. Damit wurde der Treiber „idiotensicher“gemacht, und nur eine Option bzgl. der Transistoren auf einer Seite des Vierquadrantenstellers ausgewählt werden kann. Anders gesagt, die Polarität wird ausgewählt, indem die beiden Treibersignalen, die an den beiden Enden der Spule des Motors anliegen, genutzt werden.

Beachte: Verwechslen Sie nicht RC PWM Signal und gewöhnliche PWM Signale.

Übung

Das Board der Motoren des HomeLab erlaubt den Anschluss von bis zu 4 Gleichstrommotoren. Das Schaltbild und die Anleitung für die Anschlüsse können im Kapitel „Motormodul“gefunden werden. Für jeden Motor gibt es einen Vierquadrantensteller, welcher über zwei digitale Output-Pins des Mikrocontrollers kontrolliert wird, da der *enable* Pin konstant „high“ist.Wenn beide Kontrollpins den gleichen Wert haben, wird der Motor gestoppt. Wenn verschiedene Werte vorliegen, wird der Motor in eine bestimmte Richtung gedreht. Der Status des Vierquadrantenstellers ist in der folgende Tabelle beschrieben:

Input A	Input B	Output A	Output B	Result
0	0	-	-	Der Motor wird gestoppt
1	1	+	+	Der Motor wird gestoppt
1	0	+	-	Der Motor dreht in Richtung 1
0	1	-	+	Der Motor dreht in Richtung 2

Gleichstrommotoren können durch Änderung der dazugehörigen Treiberpins mit dem Mikrocontroller kontrolliert werden. Die speziellen Funktionen um den Motor zu steuern sind in der HomeLab Bibliothek enthalten.

```
//
// Die Einrichtung von Pins als Treiberpins.
//
static pin dcmotor_pins[4][2] =
{
  { PIN(B, 7), PIN(B, 4) },
  { PIN(D, 1), PIN(D, 0) },
  { PIN(D, 7), PIN(D, 6) },
  { PIN(D, 5), PIN(D, 4) }
};

//
// Steuerung des gewählten DC Motors zulassen.
//
void dcmotor_init(unsigned char index)
{
  pin_setup_output(dcmotor_pins[index][0]);
```

```
  pin_setup_output(dcmotor_pins[index][1]);
}

//
// Festlegung der Funktion und Richtung des gewählten DC Motors.
//
void dcmotor_drive(unsigned char index, signed char direction)
{
  pin_set_to(dcmotor_pins[index][0], direction < 0);
  pin_set_to(dcmotor_pins[index][1], direction > 0);
}
```

Mit dem Array *dcmotor_pins* aus der Bibliothek, werden die Kontrollpins der vier Motorcontroller bestimmt. Bevor der Motor gesteuert wird, muss die Funktion *dcmotor_init* mit der Nummer des Motorcontrollers (0-3) benannt werden. Sie setzt die Pins als Output. Für die Steuerung ist die Funktion *dcmotor_drive* vorhanden, der negative *direction* Parameter gibt die Richtung der Motordrehung an, die andere Richtung wird mit dem positiven Parameter angegeben. 0 dient dazu, den Motor anzuhalten.

Das folgende Beispielprogramm steuert den ersten und zweiten Gleichstrommotor so, dass sie jede Sekunde ihre Drehrichtung ändern. Die Geschwindigkeit kann gesteuert werden, wenn ein Kontrollpin mit einem PWM Signal moduliert wird.

```
//
// Testprogramm für den DC Motor aus dem DC Motorenmodul des HomeLab.
//
#include <homelab/module/motors.h>
#include <homelab/delay.h>

//
// Hauptprogramm
//
int main(void)
{
  // Richtungsvariable
  signed char direction = 1;

  // Einrichtung von Motor 1 und 2.
  dcmotor_init(0);
  dcmotor_init(1);

  // Endlosschleife
  while (true)
  {
    // Ein Motor dreht sich in die eine, der andere in die
    entgegengesetzte Richtung.
    dcmotor_drive(0, -direction);
    dcmotor_drive(1, +direction);
```

```
    // Unterbrechung für 1 Sekunde.
    sw_delay_ms(1000);

    // Umkehr der Richtung.
    direction = -direction;
  }
}
```

6.6.2 Servomotor

Notwendiges Wissen: *Motor Module,* *Sensors Module,* *Digital Inputs/Outputs,* *Motors,* *Analog to Digital Converter*

Theorie

Abbildung 6.31: RC Servomotor

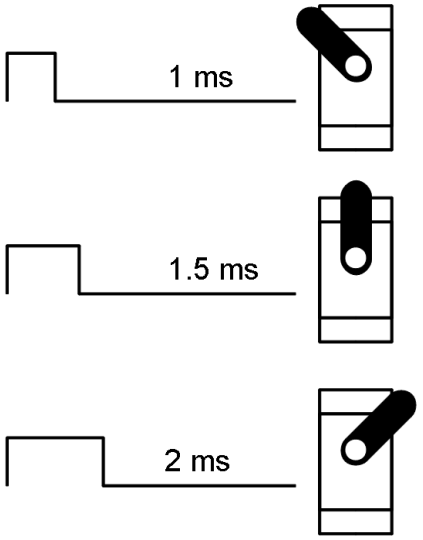


Abbildung 6.32: Das Verhältnis von Signalweite und Position der PWM des Servomotors.

Servomotoren werden oft in funkgesteuerten (RC) Modellen benutzt und sind sehr nützlich für kleine robotischen Anwendungen, da sie sehr kompakt und günstig sind.Ein RC-Servomotor verfügt über einen eingebauten Gleichstrommotor, ein Getriebe, einen Positionsfeedbacksensor (meistens ein Potentiometer, und eine Steuerelektronik. RC-Servomotren können mit einem externen PWM-Signal gesteuert werden. Wenn das Signal den RC Servotiming-Vorraussetzungen entspricht, „beschreibt"es normalerweise ein Positionsinput für die Servo-Steuer-Elektronik. Die Servomotor Elektronik vergleicht die Achsenstellung mit der eingegebenen Position und versucht eine übereinstimmende Position zu erreichen. Das Positionskontrollsignal ist eine durchgehende Rechteckschwingung, wie in dem Bild dargestellt.

RC (*radio-controlled*) Servomotoren sind in der Robotik und im Modellbau sehr verbreitete Antriebe. Sie bestehen aus einem kleinen Gleichstrommotor, einer Übersetzung und der logischen Kontrolleinheit. Normalerweise versucht der Rotor des Servomotors sich in eine bestimmte Position zu bringen und diese zu halten. Die Position des Rotors hängt von dem Kontrollsignal ab, dass der Servomotor empfangen hat. Je nach Typ des Motors kann der maximale Drehwinkel des Motors variieren. Servomotoren, die sich konstant drehen, sind selten. In diesem Fall gibt das Kontrollsignal nicht den Rotationswinkel, sondern die Rotationsgeschwindigkeit an. Servomotor-Hacks sind auch sehr verbreitet. Hierdurch wird ein positionsbestimmender Servomotor zu einem konstant drehenden. In diesem Fall wird das Feedback Potentiometer durch zwei Widerstände ersetzt, und der mechanische Widerstand, welcher verhindert, dass eine vollständige Rotation stattfinden kann, wird entfernt.Eine wichtige Eigenschaft des Servomotors ist das Kraft-Gewicht-Verhältnis.

Das Kontrollsignal des Servomotors ist ein speziell pulsweitenmoduliertes Signal (PWM), bei dem die Pulsweite die Position des Rotors angibt. Die Periode des Signals liegt bei 20 ms (50 Hz) und die Weite der hohen Periode ist 1 ms - 2 ms. 1 ms markiert die eine äußerste Position 2 ms die Andere. 1,5 ms markieren die Mittelposition des Servomotorrotors.

Traditionelle RC-Servomotoren sind auch bekannt als analoge Servomotoren. Dies liegt daran, dass in dem letzten Jahrzehnt so genannte digitale Servomotoren auf den Markt kamen. Der Unterschied zwischen beiden besteht darin, dass analoge RC-Servomotoren mit einem 50 Hz PWM Signal gesteuert werden, digitale RC-

Servomotoren durch einen Mikrocontroller mit einem höherfrequenten Signal. Das Inputsignal ist das Gleiche, jedoch ist durch die höhere Modulationsfrequenz wird eine schnellere und präzisere Positionsbestimmung ermöglicht.

Übung

Auf dem Platine des HomeLab Motormoduls sind 2 Anschlüsse für RC-Servomotoren vorhanden.Die PWM-Enden der Stecker sind an den Pins PB5 und PB6 am Mikrocontroller angeschlossen, dessen sekundäre Funktion das Vergleichen der Einheiten A und B von Timer 1 ist. Timer 1 kann ein PWM Signal erzeugen, und daher ist das Steuern eines Motors sehr einfach zu programmieren. Die einzige Schwierigkeit besteht im Einstellen des Timers.

Timer 1 muss in den PWM-Produktionsmodus eingestellt werden, wobei der maximale Wert des Timers mit dem ICR Register bestimmt wird. Durch Änderung des maximales Wertes im Programm und im Taktgeber des Timers, kann die präzise PWM Frequenz zur Steuerung des Servomotors bestimmt werden. Mit dem Vergleichs-Register des Timers werden die Längen für beiden hohen Semi-Perioden des Signals bestimmt. Der Timer hat eine spezielle Vergleichseinheit, welche den Wert des Zählwerks beobachtet und für den Fall, dass es gleich dem Wert des Vergleichs-Registers ist, den Output-Wert der Vergleichseinheit ändert. Es folgt ein Programm-Code der Servomotor-Steuerbibliothek des HomeLab. Für die Ausführung nutzt es Parameter für Timer welche durch Makrofunktionen bestimmt werden. Zum Beispiel, wird die Periode durch die F-CPU Konstante angegeben, welche die Taktrate des Mikrocontrollers angibt. Wenn man Makros nutzt, müssen die Parameter des Timers für verschiedene Taktzyklen nicht berechnet werden und der Compiler konvertiert die Operationen mit Makros zu Konstanten. Dadurch erhöht sich der Programmspeicher nicht, und es wird auch nicht mehr Rechenzeit benötigt.

```
//
// Der WErt des Timers (20 ms) zum ERreichen der vollen PWM-Periode.
// F_CPU ist die Taktrate des Mikrocontrollers, welche durch 50 Hz und
    8 dividiert wird.
//
//
#define PWM_PERIOD       (F_CPU / 8 / 50)

//
// Mittlere Position der PWM des Servomotors (5 ms / 20 ms)
// Mittlere Position liegt bei 15/200 der vollen Periode.
//
#define PWM_MIDDLE_POS  (PWM_PERIOD * 15 / 200)

//
// Faktor zur Konvertierung von Prozenten in Perioden(-100% to 100%).
// +1 wird addiert um sicherzustellen, dass die Halbperioden die
    Grenzen von 1 ms und 2 ms erreichen oder //  etwas höher liegen.
//
#define PWM_RATIO        (PWM_PERIOD / 20 / 2 / 100 + 1)

//
```

```
// Einrichtung der Pins.
//
static pin servo_pins[2] =
{
  PIN(B, 5), PIN(B, 6)
};

//
// Servomotor vorbereiten.
//
void servomotor_init(unsigned char index)
{
  // Pin des PWM Signals als Output.
  pin_setup_output(servo_pins[index]);

  // Einrichtung von Timer 1.
  // Vorzähler = 8
  // Schneller PWM Modus, wobei TOP = ICR
  // OUTA und OUTB auf low im Vergleich.
  timer1_init_fast_pwm(
    TIMER1_PRESCALE_8,
    TIMER1_FAST_PWM_TOP_ICR,
    TIMER1_FAST_PWM_OUTPUT_CLEAR_ON_MATCH,
    TIMER1_FAST_PWM_OUTPUT_CLEAR_ON_MATCH,
    TIMER1_FAST_PWM_OUTPUT_DISABLE);

  // Festlegung der Periode durch maximalen Wert.
  timer1_set_input_capture_value(PWM_PERIOD);
}

//
// Festlegung der Postition des Servomotors.
// Parameter der Position liegen zwischen -100% und +100%.
//
void servomotor_position(unsigned char index, signed short position)
{
  switch (index)
  {
    case 0:
      timer1_set_compare_match_unitA_value(
        PWM_MIDDLE_POS + position * PWM_RATIO);
      break;

    case 1:
      timer1_set_compare_match_unitB_value(
```

```
        PWM_MIDDLE_POS + position * PWM_RATIO);
      break;
  }
}
```

Das Beispielprogramm nutzt beschriebene Funktionen der HomeLab Bibliothek. Am Anfang des Programms wird der erste PWM-Signal Generator des Servomotors mit der *servomotor_init* Funktion gestartet. Der Wert der Position des Servomotors wird durch die Kanalnummer 3 des Analog-Digital-Konverters (ADC) empfangen, an dem ein Potentiometer an der Sensorplatine angeschlossen ist. Um die Reichweite von -100 % bis +100 %, die für die Steuerung des Servomotors notwendig, ist zu erhalten wird die Hälfte des Maximums (512) vom ADC Wert abgezogen und das Ergebnis durch fünf dividiert. Das Ergebnis ist +/- 102, kleine Ungenauigkeiten können unbeachtet bleiben, da Servomotoren sich auch in der Relation des PWM-Signals und des Drehwinkels unterscheiden. Die finale PWM Halbperiodenweite in Anwendungen muss über das „Try-and-Error"-Verfahren gefunden werden. Auch wenn die Funksteuerungen von RC-Modellen Möglichkeiten für ein präzises Setup besitzen. Wenn das Programm läuft, wird die Rotorposition des Servomotors je nach Position des Potentiometers verändert.

```
//
// Testprogramm des Motormoduls des HomeLab kit.
//
#include <homelab/adc.h>
#include <homelab/module/motors.h>

//
// Hauptprogramm.
//
int main(void)
{
  short position;

  // Einrichtung des ADC.
  adc_init(ADC_REF_AVCC, ADC_PRESCALE_8);

  // Einrichtung des Motors.
  servomotor_init(0);

  // Endlosschleife.
  while (true)
  {
    // Auslesen der Position aus dem Potentiometer und Konvertiereung
    der Reichweite
    // des Servomotors.
    position = ((short)adc_get_value(3) - (short)512) / (short)5;

    // Festlegung der Position des Servomotors.
    servomotor_position(0, position);
  }
```

```
}
```

6.6.3 Schrittmotor

Notwendiges Wissen: *Motor Module,* *Digital Inputs/Outputs,* *Motors*

Theorie

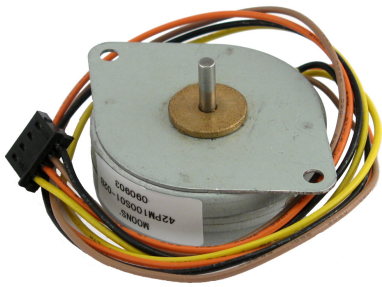


Abbildung 6.33: Schrittmotor

Schrittmotoren können generell in unipolare und bipolare Schrittmotoren eingeteilt werden.Unipolare Schrittmotoren haben charakteristische Windungen welche die zwei Spulen in vier unterteilen. Schrittmotoren haben keine eingebauten Bürsten oder Elektronik, d.h. jegliche Kommunikation muss extern erfolgen. Der am meisten benutzte Kommutationstyp ist der offene-Regelkreis- („open-loop“) Modus: der Der Schrittmotortreiber lädt die Spulen nach einem bestimmten Muster, nutzt aber kein Feedback. Schritte können so verpasst werden falls die Motorwelle eine Drehmomentüberladung erfährt. Verpasste Schritte verursachen ungenaue Positionierung. Bipolare Schrittmotoren haben normalerweise vier Drähte und zwei separate Spulen im Inneren. Sie ähneln den unipolaren Schrittmotoren stark. Unipolare Schrittmotoren können als bipolare Schrittmotoren genutzt werden, aber nicht umgekehrt.

Schrittmotoren werden oft in Applikationen genutzt, die Genauigkeit benötigen. Anders als Gleichstrommotoren haben Schrittmotoren weder Bürsten, noch Kommutatoren. Sie haben verschiedene unabhängige Spulen, welche durch externe Elektronik (Treiber) angetrieben werden. Um den Rotor anzutreiben, werden die Spulen Schritt für Schritt umgeschaltet, ohne Feedback. Das ist der Nachteil des Schrittmotors. Bei mechanischer Überladung, wenn der Rotor nicht rotiert, werden so die Schritte durcheinander gebracht und die Bewegung wird ungenau. Zwei Typen von Schrittmotoren werden anhand der Spulen unterschieden: Unipolare und bipolare Schrittmotoren. Durch Aufbau werden drei zusätzliche Segemente bedacht:

```
* Reluktanzschrittmotor (hohe Genauigkeit, wenig Drehmoment, niedriger
    Preis)
```

```
* Permanentmagnetschrittmotor (niedrige Genauigkeit,hohes Drehmoment,
    niedriger Preis)
* Hybrider Schrittmotor (hohe Genauigkeit, hohes Drehmoment, hoher
    Preis)
```

Der Reluktanzschrittmotor verfügt über gezahnte Windungen und einen gezahnten Weicheisenrotor. Die größte Zugkraft entsteht, wenn die Zähne beider Seiten sich gegenseitig abdecken. Der Permanentmagnetschrittmotor bestitz, wie der Name schon sagt, Permanentmagneten welche sich je nach Polarität der Windung orientieren. Der hybride Schrittmotor nutzt beide Techniken.

Je nach Modell des Schrittmotors, braucht eine vollständige Rotation (360°) des Rotors hunderte Schritte von Stromwendungen. Für stabile und weiche Bewegungen, wird eine bestimmte Kontrollelektronik genutzt, welche den Motor je nach Parameter kontrolliert (Trägheit des Rotors, Drehmoment, Resonanz etc.). Zusätzlich zur Kontrollelektronik werden verschiedene Methoden zur Stromwendung genutzt. Wird eine Windung in Reihe umgewandelt, wird dieser Vorgang „Ganzschrittantrieb“genannt, verändert sich der Antrieb zwischen zwei Wicklungen, nennt man diesen Halbschritt. Darüber hinaus werden „Kosinus-Mikroschritte“genutzt, um eine sehr genaue und weiche Kontrolle zu ermöglichen.

Unipolare Schrittmotoren

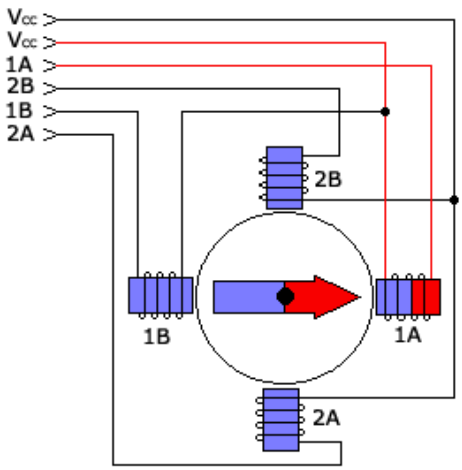


Abbildung 6.34: Die Wicklungen eines unipolaren Schrittmotors

Unipolare Schrittmotoren haben fünf oder sechs Anschlüsse. Je nach Aufbau des Motors werden nur $^1/_4$ der Wicklungen aktiviert. *Vcc* Leitungen werden normalerweise an der positiven Stromquelle angeschlossen. Während der Umwandlung sind die Enden der Wicklungen 1a, 1b, 2a und 2b über Transistoren (Transistorarray der Motorplatine ULN2803) nur an die Masse angeschlossen, was die Kontrollelektronik recht einfach macht.

Bipolare Schrittmotoren

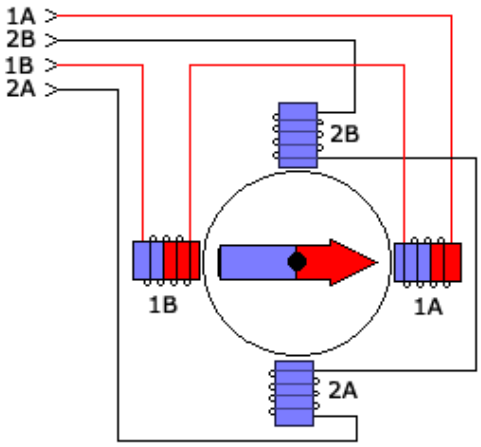


Abbildung 6.35: Die Wicklungen eines Bipolaren Schrittmotors.

Bipolare Schrittmotoren unterscheiden sich von unipolaren, da sie die Polarität ihrer Wicklungen während der Umwandlung verändern. Die Hälfte der Wicklungen wird gleichzeitig aktiviert, daher sind sie effizenter als unipolare Schrittmotoren. Bipolare Schrittmotoren haben 4 Anschlüsse, jeder wird an einen anderen Vierquadrantensteller angeschlossen (Treiber L293 auf der Motorenplatine). Während der Kommutation legen die Vierquadrantensteller entweder negative oder positive Spannung am Ende der Wicklung an. Unipolare Motoren können mit einem bipolaren Treiber gestartet werden indem einfach die Leitungen 1a, 1b, 2a und 2b der Wicklungen angeschlossen werden (*Vcc* wird nicht angeschlossen).

Die notwendige Kommutation, um Schrittmotoren mit Wicklungen im Ganzschrittmodus und im Halbschrittmodus zu kontrollieren, wird in der Tabelle unhalb angezeigt. Da in Treibern für unipolare Schrittmotoren nur das Öffnen von Transistoren stattfindet, werden die Schritte mit 0 und 1 markiert. Zur Steuerung eines bipolaren Schrittmotors braucht es Signale und daher wird jeder Schritt mit der Polarität des Treiber-Outputs markiert:

	Unipolar				**Bipolar**			
Schritt	**1A**	**2A**	**1B**	**2B**	**1A**	**2A**	**1B**	**2B**
Ganzschritt								
1	**1**	0	0	0	**+**	-	-	-
2	0	**1**	0	0	-	**+**	-	-
3	0	0	**1**	0	-	-	**+**	-
4	0	0	0	**1**	-	-	-	**+**
Halbschritt								
1	**1**	0	0	0	**+**	-	-	-
2	**1**	**1**	0	0	**+**	**+**	-	-
3	0	**1**	0	0	-	**+**	-	-
4	0	**1**	**1**	0	-	**+**	**+**	-
5	0	0	**1**	0	-	-	**+**	-
6	0	0	**1**	**1**	-	-	**+**	**+**
7	0	0	0	**1**	-	-	-	**+**
8	**1**	0	0	**1**	**+**	-	-	**+**

Übung

Das Ziel dieser Aufgabe ist es, unter Anwendung der oben beschrieben Methode einen bipolaren Schrittmotor zu starten, welcher mit einem unipolaren Schrittmotor getauscht werden kann. Es gibt Treiber auf der Motorenplatine, welche vom Mikrocontroller über vier Inputpins kontrolliert werden. Jeder Pin gibt die Polarität einer Wicklung wieder. Die Spannung am Ende der Wicklung ist positiv wenn der Pin high ist und negativ wenn der Pin low ist.Zu den Enden 1A, 1B, 2A und 2B gehören die entsprechenden Pins PB0, PB1, PB2 Und PB3 am Mikrocontroller.

In der HomeLab Bibliothek gibt es die Funktion *bipolar_init*, um bipolare Schrittmotoren zu steuern, indem die Pins als Output gesetzt werden. Die Funktion *bipolar_halfstep* führt Rotationen mit bestimmten Halbschritten aus. Die Kommutation wird mit der Tabelle der Halbschritte durchgeführt, aber es werden komplexere Bitoperationen genutzt.

```
//
// Vorbereitung der Steuerung des bipolaren Schrittmotors.
//
void bipolar_init(void)
{
    DDRB |= 0x0F;
    PORTB &= 0xF0;
}

//
// Bewegung des bipolaren Schrittmotors mit Halbschritten.
//
void bipolar_halfstep(signed char dir,
    unsigned short num_steps, unsigned char speed)
{
    unsigned short i;
    unsigned char pattern, state1 = 0, state2 = 1;

    // Festlegung der Richtung +- 1
    dir = ((dir < 0) ? -1 : +1);

    // Durchführung von Halbschritten.
    for (i = 0; i < num_steps; i++)
    {
        state1 += dir;
        state2 += dir;

        // Erstellung des Musters.
        pattern = (1 << ((state1 % 8) >> 1)) |
                  (1 << ((state2 % 8) >> 1));

        // Output setzen.
        PORTB = (PORTB & 0xF0) | (pattern & 0x0F);
```

```
        // Pause um auf die Ausführung des Schrittes zu warten.
        sw_delay_ms(speed);
    }

    // Anhalten des Motors.
    PORTB &= 0xF0;
}
```

Die Funktion wird durch ein Beispielprogramm demonstriert, welches den Motor abwechselnd alle 200 Halbschritte in eine Richtung rotieren lässt. Die Geschwindigkeit der Rotation wird durch die Länge der Pausen zwischen den Schritten bestimmt. Wenn die Pause zu kurz ist, kann der Motor sich aufgrund der Trägheit nicht drehen.

```
//
// Das Testprogramm für den bipolaren Schrittmotor des Motormoduls
// des HomeLab.
//
#include <homelab/module/motors.h>

//
// Hauptprogramm.
//
int main(void)
{
    // Einrichtung des Motors.
    bipolar_init();

    // Endlosschleife.
    while (true)
    {
        // Bewegung des Rotors 200 Halbschritte in eine Richtung mit
    einer Geschwindigkeit von 30 ms/Schritt.
        bipolar_halfstep(+1, 200, 30);

        // Bewegung des Rotors 200 Halbschritte in die andere Richtung
    mit einer Geschwindigkeit von 30 ms/Schritt.
        bipolar_halfstep(-1, 200, 30);
    }
}
```

6.6.4 Aufgaben

Ziel ist es, ein Programm zu schreiben, welches die folgenden Aufgaben erfüllen kann.

Aufwärmübung

Steuern des Gleichstrommotors mit dem digitalen Board. Durch Betätigung von Schalter S1 leuchtet LED1 auf und der Motor dreht sich im Uhrzeigersinn. Durch Betätigung von Schalter S3 leuchtet die LED3 auf und der Motor dreht sich gegen den Uhrzeigersinn. Durch Betätigung von Schalter S2 leuchtet LED2 auf und der Motor wird gestoppt.

Für Anfänger

1. Roboter: Durch Nutzung von zwei Gleichstrommotoren und Berührungssensoren wird die Bewegung eines Roboters simuliert. Berührungssensoren sind die Schalter am digitalen Board (S1…S3). Die Motoren werden durch die Betätigung der Schalter gesteuert. S1 stoppt den linken Motor für zwei Sekunden und startet dann beide Motoren mit voller Geschwindigkeit. S2 stoppt den rechten Motor für zwei Sekunden und startet dann beide Motoren mit voller Geschwindigkeit. Wenn beide Schalter betätigt werden drehen beide Motoren rückwärts, bis der Schalter losgelassen wird.

2. Servomotor: Der Servomotor wird mit den Schaltern des digitalen Boards kontrolliert. Wenn S1 gedrückt wird macht der Servomotor einen Schritt nach rechts, bei S3 einen Schritt nach links und S2 bewegt den Servomotor wieder in die ursprüngliche Position (Mitte). Die Position des Motors wird direkt auf dem 7-Segment Display dargestellt (jede Nummer steht für 10° Drehung, Mittelposition = 5)

3. Radar: Die Funktion eines Radars wird simuliert. Um ein Objekt, welches näher als 0.5 Meter ist, zu identifizieren, wird auf dem Hebel des Servomotors der IR-Entfernungsmesser installiert. Der Hebel des Servomotors bewegt sich konstant von einem Ende zum anderen Ende der Bewegungsreichweite des Motors. Falls ein Objekt näher als 0.5 Meter am Sensor ist, wird der Servomotor für 5 Sekunden gestoppt und eine LED (PB7) zeigt auf dem Controllerboard die Entdeckung eines Objekts an.

4. Schrittmotor: Nach jeder Betätigung von S1 oder S3, rotiert der Motor 10 Schritte im oder gegen den Uhrzeigersinn. Die Rotation wird sofort angehalten wenn S2 gedrückt wird.

5. Alle drei Motorentypen werden angeschlossen. Durch Betätigung eines Schalters wird ein bestimmter Motor gestartet und angehalten. S1 steuert den Gleichstrommotor, S2 den Servomotor und S3 den Schrittmotor.

Für Fortgeschrittene

1. Der Gleichstrommotor beschleunigt wenn S1 gedrückt wird und hält die momentane Geschwindigkeit, wenn der Schalter losgelassen wird. Wenn S2 gedrückt wird, erfährt der Motor eine gleichmäßige Abbremsung. Wenn S3 gedrückt wird, hält der Motor sofort an (Simulation eines Notstopps).

2. Ein Objekt verfolgen: Durch das Nutzen des Ultraschall-Entfernungsmessers, welcher auf dem Hebel des Servomotors installiert wird, verfolgt der Servomotor ein vorbeikommendes Objekt. Der Motor dreht sich je nach Bewegung des Objekts, so das immer das Objekt in der Mitte des Mess-Sektors des Sensors ist.

3. der Schrittmotor hält die letzte Position des Motors nach Änderung jeder Sequenz. Wenn eine neue Sequenz aktiviert wird, nutzen Sie eine Variable, so dass die Bewegung exakt von der letzten Position des Motors fortgeführt wird.

4. Beschleunigung: Das Programm ermöglicht die Beschleunigung oder Abbremsung des Schrittmotors. Nutzen Sie lineare Geschwindigkeitssteigerungen welche bei visueller Betrachtung einfach identifiziert werden können. Weite Bewegungen müssen folgendes Schema erfüllen: Beschleunigung ->konstante Geschwindigkeit ->Abbremsung.

5. Entwerfen Sie einen PID-Regulator für einen Gleichstrommotor. Achtung! Diese Aufgabe benötigt einen Motor mit Feedback. Diese Aufgabe kann daher auch theoretisch gelöst werden.

6.6.5 Fragen

1. Welcher Vierquadrantensteller wird genutzt? Nach welchem Prinzip funktioniert er?

2. Wie wird die Position der Achse eines RC-Servomotors bestimmt?

3. Was ist der Hauptunterschied zwischen unipolaren und einem bipolaren Schrittmotoren?

4. Wie können Halbschritt und Mikroschrittmodi eines Schrittmotors genutzt werden? Nennen Sie ein Beispiel.

5. Wie kann die Rotationsgeschwindigkeit eines DC Motors gesteuert werden? Nennen Sie ein Beispiel.

6. Welcher PWM-Arbeitszyklus wird benötigt um eine Gleichstrommotorrotation mit 70% der nominalen Geschwindigkeit zu erreichen?

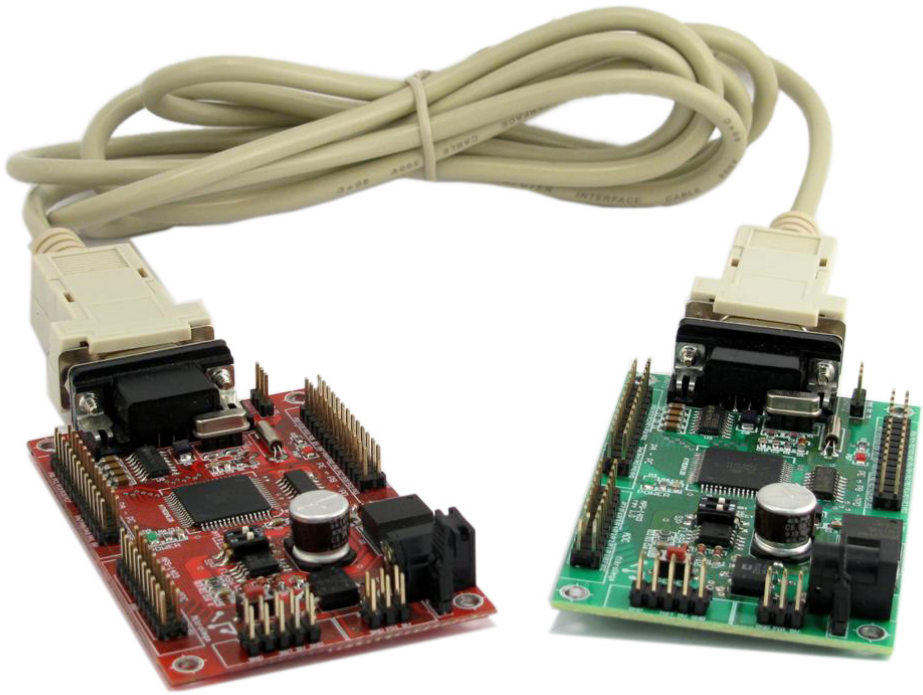
7. Wie wird die Richtung der Motorrotation bestimmt, wenn ein Encoder genutzt wird?

8. Wie kann ein Gleichstrommotor elektronisch gebremst werden?

9. Was passiert, wenn das Schema der Stromwendung eines Schrittmotors sich zu schnell ändert?

10. Ist dynamisches Bremsen möglich? Wenn ja, wie?

6.7 Datenschnittstellen



Mit Microcontrollern können Antriebe gesteuert werden, Werte von Sensoren ausgelesen werden und viel mehr. Diese Geräte müssen dazu jedoch immer mit dem Mikrocontroller verbunden sein; es ist somit nicht möglich, durch einfache Signale zu kommunizieren. Das liegt darin begründet, dass zu viele Steuerungssignale zur Steuerung der Schnittstelle benötigt werden oder, dass zu viele Daten gesendet werden müssen. Daher wurden zahlreiche Standards von Datenschnittstellen für Mikrocontroller oder andere Elektronik entwickelt. Die Standards bestimmen die elektrischen Parameter der Signale und die Übertragungsregeln für die Signale (das Protokoll).

Ein einfaches Beispiel eines Protokolls ist der Morse Code, womit Informationen durch Nutzung von Pieptönen und Pausen in verschiedenen Abständen übertragen werden. Digitale Datenübertragungsprotokolle arbeiten ähnlich. Dort werden die Informationen als Bit Werte gesendet und je nach Schnittstelle Angepasst an versschiedene Bedürfnisse wurden diverse Schnittstellen zur Datenübertragung entwickelt. Da die Datenmengen jedoch stetig ansteigen, werden immer wieder neue Methoden entwickelt. Bei elektronischen Komponenten ist die Situation dahingehend etwas stabiler. Die I^2C, SPI und UART Schnittstellen werden schon sehr lange verwendet. Traditionellere Schnittstellensysteme für die Übertragung innerhalb eines Systems sind RS-232, Rs-485, LIN and CAN, aber viele Mikrocontroller werden schon mit USB, Ethernet und kabellosen ZigBee Schnittstellen produziert. Dieses Kapitel konzentriert sich auf die RS-232 Schnittstelle.

6.7.1 RS-232

Notwendiges Wissen: *Controller module,* *USART,* *Serial Interface,* *Alphanumeric LCD*

Theorie

Abbildung 6.36: Cable RS-232. Der linke Stecker ist „male“, der rechte „female“.“

Die RS-232 ist ein Standard unter den physischen Datenschnittstellen und wird zur Übertragung von Binärdaten genutzt. Dieser Standard wird überwiegend für serielle Anschlüsse von Computern verwendet, welche gewöhnlich als „COM“-Anschlüsse bezeichnet werden. Inzwischen wurden die RS-232 größtenteils durch USB-Schnittstellen ersetzt. Aufgrund ihrer Einfachheit werden sie jedoch weiterhin erfolgreich in Freizeitanwendungen verwendet, insbesondere wenn USB - RS-232 Konverter vorhanden sind. Der RS-232 Standard legt die Stecker, die elektronischen Parameter sowie die Bedeutung der Signale fest, nicht aber das Protokoll.

Die RS-232 Schnittstelle wird hauptsächlich in Verbindung mit dem UART Datenübermittlungsmodul genutzt. Diese UART-Hardware verfügt über ein standardisiertes Protokoll, legt jedoch keine Stecker oder andere Dinge fest. Die RS-232 Schnittstelle erweitert somit das UART Modul. UART ist ein Peripheriemodul des Mikrocontrollers, dessen digitaler In- und Output nicht mit den elektronischen Parametern des RS-232 korrespondiert. Aus diesem Grund werden die beiden über einen speziellen Pegelkonverter miteinander verbunden. Der bekannteste Pegelkonverter für RS-232 und TLL/COMS ist der MAX232.

Die Bezeichnung UART steht für „Universal Asynchronous Receiver Transmitter“. Die USART-Schnittstelle bietet ergänzend die Möglichkeit, Daten synchron zum Taktsignal zu senden. UART kann auch als serielle Schnittstelle bezeichnet werden. Die serielle Schnittstelle ist ein Datentransfermechanismus, wobei jedes Bit einzeln übermittelt wird. Um beispielsweise 1 Byte zu senden, werden 8 Bits in einem bestimmten Intervall übermittelt. Dadurch wird an der seriellen Leitungsschnittstelle, also an einen Pin des Mikrocontrollers, der Wert der Stromspannung nach einer bestimmten Zeit verändert, zunächst ist dieser niedrig und daraufhin hoch. Normalerweise gibt es zwei Geräte die an der seriellen Schnittstelle angeschlossen werden. Das eine dient zur Übertragung von Information (durch Veränderung des Pinwertes), das andere zum Empfang (durch Erfassung des Pinwertes). Der Übertragungspin ist mit TX, der Empfangspin mit RX gekennzeichnet. Auf

einer Leitung geht der Informationsfluss immer nur in eine Richtung. Um Daten in die andere Richtung zu senden, muss eine andere Leitung genutzt werden. Werden Daten über zwei Leitungen gleichzeitig gesendet, wird dieses „Full Duplex Bus“genannt.

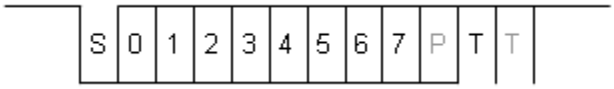


Abbildung 6.37: Der Rahmen des UART, mit S als Startbit , 0-7 als Datenbits, P als Paritätsbit (falls dieses existiert) und T als Stopbit (oder 2).

Die Datenübertragung erfolgt über den Rahmen der UART-Schnittstelle, welcher je nach Konfiguration 5 bis 9 Datenbits enthält. Am häufigsten sind 8 Bits (1 Byte). Zusätzlich zu den Datenbits werden durch den Rahmen weitere Bits übertragen, welche dazu genutzt werden, den Augenblick der Ankunft von Daten sowie des Abschlusses der Übertragung auf der Empfängerseite zu erkennen. Ersteres wird Startbit genannt und hat immer den Wert 0. Das zweite ist das Stopbit, welches immer den Wert 1 hat. Vor dem Stopbit gibt es in einigen Fällen noch das Paritätsbit. Dieses dient zur Kontrolle der Richtigkeit. Das Paritätsbit zeigt an, ob die Anzahl der Einsen in den Datenbits gerade oder ungerade ist. Die Art der Ablesung hängt von der Konfiguration der UART Schnittstelle ab. Das Paritätsbit wird aber normalerweise nicht mehr verwendet und kann im Rahmen der Konfiguration unterdrückt werden. So wie das Paritätsbit konfiguriert werden kann, kann auch die Anzahl von Daten- und Stopbits variiert werden.

Zusätzlich zur Rahmenstruktur gibt es einen weiteren wichtigen Parameter - die *Baudrate*, wodurch die Anzahl der in einer Sekunde übertragenen Symbole festgelegt wird. Die Baudrate zeigt die Anzahl der Symbole an. Bei der Verwendung von UART entspricht 1 Baud einem Bit, weshalb im Zusammenhang mit dem Rahmen von Bits gesprochen wurde. Allgemein ist es egal, welche Baudrate zur Datenübertragung genutzt wird, es gibt jedoch eine einige häufig verwendete Baudraten, welche verwendet werden sollten. Diese sind zum Beispiel: 9600 bps, 19200bps, 38400 bps, 57600 bps, 115200 bps.

Darüber hinaus ist es wichtig zu wissen, dass der RS-232 Standard zusätzlich zu den Datensignalen (RX, TX) auch über Datenflusskontrollpins DTR, DCD, DSR, RI, RTS undCTS verfügt, welche zur Kontrolle der Kommunikation zwischen den Geräten dienen. Sie können zum Beispiel genutzt werden, um festzustellen ob das Gerät zum Empfang von Daten bereit ist oder nicht. Da die RS-232 Schnittstelle ursprünglich darauf abzielt, Computer an ein Modem anzuschließen, können (konnten) einige Signale dazu genutzt werden, um den Status von Telefonleitungen anzuzeigen.

Praktisches Beispiel

Die Platine des Controllermoduls ist mit einem RS-232 male-Stecker ausgerüstet. Darüber kann der Controller an einen Computer oder einen anderen Controller angeschlossen werden. Um ihn an einen Computer anzuschließen, muss ein normales nicht invertiertes Kabel genutzt werden, dessen eines Ende male und anderes Ende female ist. Um den Controller an einen anderen Controller anzuschließen, muss ein Kabel verwendet werden, bei dem RX, TX sowie Kontrollsignale für die Stromstärke senkrecht invertiert sind und beide Stecker female sind. Das invertierte Kabel wird auch Nullmodemkabel genannt. Nachfolgend ist ein Beispielprogramm zur Nutzung der UART Schnittstelle dargestellt. Wird das Programm gestartet, übermittelt es ein „Willkommen“über die RS-232 Schnittstelle und zeigt empfangene Nachrichten an. Es werden das LCD-Display und die USART Bibliotheken genutzt.

```
//
// Anschluss des Controllermoduls des HomeLab an einen Computer über
    die RS-232 Schnittstelle.
// Im Beispiel wird ein digitales Input-Output-Modul mit LCD Display
    verwendet.
// Der im Terminal des Computers eingegebene Text wird auf dem LCD
    Display angezeigt.
//
#include <homelab/usart.h>
#include <homelab/module/lcd_alpha.h>

//
// Festlegung der USART Schnittstelle.
//
usart port = USART(0);

//
// Hauptprogramm
//
int main(void)
{
  char c;
  unsigned char row = 1;

  // Installation der USART Schnittstelle.
  usart_init_async(port,
    USART_DATABITS_8,
    USART_STOPBITS_ONE,
    USART_PARITY_NONE,
    USART_BAUDRATE_ASYNC(9600));

  // Installation des LCD.
  lcd_alpha_init(LCD_ALPHA_DISP_ON_BLINK);

  // Anzeige eines Begrüßungstextes auf dem Display.
  lcd_alpha_write_string("Waiting for the message");

  // Cursor an den Beginn der zweiten Reihe setzen.
  lcd_alpha_goto_xy(0, row);

  // "Hallo" zum Computer sagen.
  usart_send_string(port, "Hello, write something!\r\n");

  // Endlosschleife
  while (true)
```

```
  {
    // Zeichen aus der seriellen Schnittstelle lesen.
    if (usart_try_read_char(port, &c))
    {
      // Befassen wir uns mit dem Zeichen zur Änderung der Reihe?
      if (c = '\r')
      {
        // Änderung der Reihe.
        row = 1 - row;

        // Vorherige Nachricht aus der Reihe löschen.
        lcd_alpha_clear_line(row);
      }
      else
      {
        // Ausgabe des Zeichens auf dem Display.
        lcd_alpha_write_char(c);
      }
    }
  }
}
```

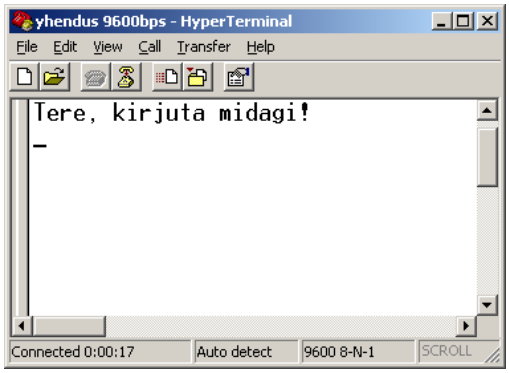


Abbildung 6.38: Das Fenster des HyperTerminal

Windows XP OS beinhaltet das Programm HyperTerminal. Dieses Programm kann über das *Start* Menü geöffnet werden, indem *Zubehör* → *Kommunikation* → *HyperTerminal* ausgewählt wird. Wählen Sie zur Konfiguration 9600 bps aus, 1 Startbit und 1 Stopbit ohne Paritäts- und Datenstromkontrolle. Wenn HyperTerminal geöffnet ist, während der Mikrocontroller startet, erscheint ein Begrüßungstext auf dem Display. Die über das Fenster eingegebenen Buchstaben werden auf dem alphanumerischen LCD dargestellt. Durch Betätigung der *Enter* Taste wird die Reihe auf dem LCD gewechselt.

6.7.2 ZigBee

— MISSING PAGE —— MISSING PAGE —

6.7.3 Aufgabe

Ziel ist es ein Programm zu schreiben, welches die folgenden Aufgaben erfüllt.

Aufwärmübung

Es wird eine Nummer an den Computer gesendet, diejede Sekunde größer wird. Die Nummer muss in Textform sein und mit einem Reihenänderungszeichen enden (\n).

Für Anfänger

1. Es wird auf die Befehle (in Buchstaben), welche über die RS-232 Schnittstelle von einem Computer gesendet werden, gewartet. Der Befehl „R“lässt die grüne LED leuchten, „K“die gelbe und „P“die rote.

„R“schaltet alle LEDs aus.

1. Wenn ein Schalter betätigt wird, wird der Name dieses Schalters (S1, S2, S3) am Computer über dieRS-232 Schnittstelle empfangen.

Für Fortgeschrittene

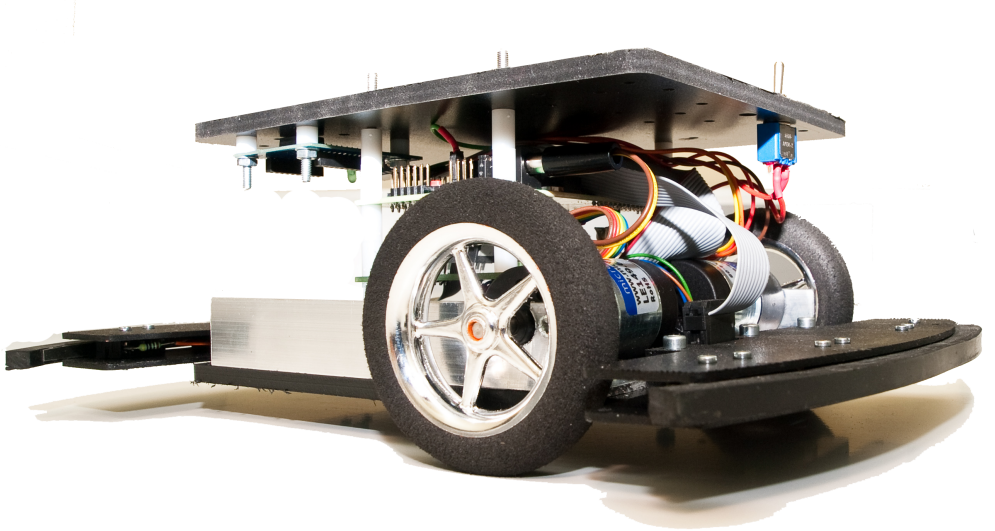
1. Kommunikation zwischen zwei RS-232 Controllern. Wird ein Schalter betätigt, sendet der eine Controller dem Anderen die Nummer des Schalters. Wenn die Nummer empfangen wurde, wird der Status der zugehörigen LED verändert.'1' $\rightarrow$ grün, '2' $\rightarrow$ gelb, '3' $\rightarrow$ rot. Benötigt 2 Controller, die Software ist jedoch für beide identisch.

2. Erstellen Sie ein „Teletype“Gerät, welches Änderungen von Textnachrichten zwischen zwei Controllern über die RS-232 Schnittstelle ermöglicht. Nutzen Sie das LCD um die Nachrichten anzuzeigen. In der ersten Reihe muss die eingegebene Nachricht zu sehen sein, in der zweiten Reiche die zuletzt empfangene Nachricht. Um eine Nachricht einzugeben können Potentiometer und / oder Schalter genutzt werden.

6.7.4 Fragen

1. Beschreiben Sie das UART Paket.
2. Was ist die Baud-Rate?
3. Was ist der Unterschied zwischen Voll/Halb-Duplex?
4. Finden Sie mindestens 5 Sensoren, welche die serielle Schnittstelle nutzen.
5. Nennen Sie den Unterschied zwischen UART und USART Schnittstelle. Welche ist schneller?
6. Wie funktioniert die SPI Schnittstelle?
7. Nennen Sie Schnittstellen, die das Anschließen von mind. 100 Geräten an einen einzigen Bus erlauben.
8. Nennen Sie verschiedene Archtiekturen und erklären Sie ihre Unterschiede.
9. Auf welchem Spannungslevel finden RS-232 und UART Verbindungen statt?
10. Wie lange benötigt, es 1MiB bei einer Baud-Rate von 9600 dps , mit 8 Daten-Bits, einem Stop-Bit und ohne Paritätskontrolle, zu senden?

7 Beispielprojekt



Das Beispielprojekt in diesem Kapitel enthält eine Anleitung zur Vorbereitung einer Projektdokumentation und erläutert, welche Themen behandelt werden sollten sowie was zu den einzelnen Themen geschrieben werden sollte. Es ist sehr hilfreich, eine Dokumentation eines typischen Projekts als Beispiel zu nehmen. Leider ist die gesamte Projektdokumentation aufgrund des Bildmaterials zu umfangreich und würde den Rahmen dieses Buches sprengen. Dennoch enthält das Beispielprojekt alle wichtigen Aspekte. So werden beispielsweise nur wichtige Details einer Zeichnung oder nur ein PCB eines elektronischen Plans gezeigt. Dennoch sollte eine tatsächliche Dokumentation Zeichnungen und Pläne für alle erstellten Details enthalten. Das Beispielprojekt soll auch die Lehrer dahin gehend unterstützen, den Schülern zu verdeutlichen, was am Ende eines Projektes erwartet wird und wie die Ergebnisse zu präsentieren sind. Je nach Situation und Lehrniveau können jedoch einige Punkte der Dokumentation ausgelassen werden.

7.1 Bericht

Um ein einfaches mechatronisches Gerät herzustellen, muss man die Konstruktion und Mechanik, Elektronik und Sensoren, Kontrollsystem und Software für das Gerät entwickeln.Am Ende des Projekts soll ein Bericht erstellt werden. Es ist eine Dokumentation welche mindesten die folgenden Punkte enthalten soll:

Titelseite

Zusammenfassung

Zusammenfassung (in Fremdsprache / Englisch)

Inhaltsverzeichnis

1. **Ursprüngliche Aufgabe**

2. **Systemvoraussetzungen und Einschränkungen**

3. **Modell des Systems**
   Struktur und Funktionalität eines Systems als Blockdiagramm. Zusätzlich können eine Anleitung zur Nutzung des Anwendungsfalldiagramms, ein Interfacediagramm etc. verfasst werden.

4. **Designlösungen**
   Mindestens drei verschiedene Konzeptlösungen für diese Aufgabe. Geeignet sind handgezeichnete Diagramme, Zeichnungen etc. Dokumentation mit Kommentaren.

5. **Mechanik**
   Am Besten ein 3D Modell mit Zeichnungen von wichtigen Verbindungen und Bauanweisungen. Falls eine Verbindung spezielle Anweisungen benötigt, dann auch Bauanleitungen und Zeichnungen hinzufügen. Wenn möglich eine Animation der Funktionen des Systems beifügen.

6. **Elektronik**
   Darstellung aller genutzten Module in einem Blockdiagramm (Controller, Motorenantrieb, Motor, Encoder etc. und ihre Verbindungen). Falls möglich geben Sie ein Layout für den Aufbau der Leiterplatte an.

7. **Control System**
   Kontrollalgorhytmus und Quellcode. Falls selbst-erstellte Funktionen genutzt werden, eine Tabelle der Funktionen und Parameter, usw. angeben.

8. **Gebrauchsfertige Lösung**
   Beschreibung und Bilder

9. **Wirtschaftliche Rechnung**
   Auflistung von Komponenten wie Kosten, geschätzte, benötigte Zeit etc.

10. **Projektmanagement**
    Zeitplan des Projekts, die Ressourcenbelegung, die Arbeitsteilung , Mitwirkung der Gruppe, Termine für Meetings etc.

11. **Zusammenfassung und Fazit**
    Was war schwer, was würden Sie anders machen, was wurde mit dem Projekt erreicht etc. Die Zusammenfassung umfasst eine kleine Beschreibung der Arbeit, aufgetretene Probleme werden genannt und das Ergebnis der Arbeit wird dargestellt. Darüber hinaus ist es sinnvoll, die eigene Meinung bezüglich der Notwendigkeit des Projekts hinzuzufügen. Außerdem bezüglich des Nutzens für die Projektmitgliedern bringt, dazu was in Zukunft anders gemacht werden sollte, was man gelernt hat und was den Leitern und Organisatoren des Projekts vorschlagen werden sollte.

Genutzte Quellen und Materialien

Anhang

Das folgende Beispielprojekt zeigt, wie eine Dokumentation und ein Bericht eines Projekts erstellt werden. Aufgrund des begrenzten Umfangs dieses Buches kann der Bericht leider nur in gekürzter Fassung dargestellt werden, er stellt aber immer noch die verschiedenen Aspekte einer Dokumentation dar.

7.2 Mobile Roboterplattform

Der mobile Roboter ist eine der populärsten Robotorkonstruktionen. Sehr verbreitet sind Sumoroboter, Sportroboter (Fußball, Volleyball usw.), Roboter die Rettungsoperationen simulieren (Feuerbekämpfung, Personen- oder Objektsuche) und viele andere. Es gibt auf der ganzen Welt und in Estland diverse Wettbewerbe für diese Art Roboter, es wurden sogar Standardklassen entwickelt (z. B. Sumoroboter). Alle diese Robotor nutzen eine mobile Plattform, welche zwar zum Teil unterschiedlich konstruiert ist oder andere Eigenschaften besitzt, deren Grundfunktion aber stets die gleiche ist. Diese umfasst die Steuerung von Motoren und einfacher Navigation, inklusive dem Ausweichen vor Objekten und der Bewegung zu einem festgelegten Punkt. Gewöhnlich wird der Hauptfunktion eine spezifische Funktion hinzugefügt, die auf die Voraussetzungen und Möglichkeiten des Projekts abgestimmt ist.

Nachfolgend werden eine Dokumentation eines typischen Projekts für eine mobile Roboterplattform sowie die verschiedenen Projektphasen dargestellt.

Ursprüngliche Aufgabe

Planen und konstruieren Sie eine multifunktionale mobile Roboterplattform mit einfacher Navigationsfunktion unter der Verwendung von HomeLab Komponenten. Die Roboterplattform muss eine einfach zu ändernde operationale Funktion besitzen, wenn sie mit verschiedenen Gadgets ausgerüstet wird:

Manipulator

Radar

Kamera

Der Roboter muss sich in geschlossenen Räumen auf flachen Böden bewegen können.

Voraussetzungen

Maximale Abmessung: 20 cm x 20 cm x 20 cm

maximales Gewicht: 2 kg

Höchstgeschwindigkeit: 0,2 m/s

volle Selbstständigkeit

Einschränkung

Muss überwiegend aus HomeLab Komponenten bestehen

Kostengrenze: 10000 EEK (630 €)

Grundlegendes Modell des Systems

Das grundlegende Modell des System wird mit einem Blockdiagramm dargestellt. Es beschreibt die Struktur, das Verhalten und andere wichtige Aspekte des Systems. Als Beispiel ist nachfolgend das hierarchische Modell des System abgebildet.

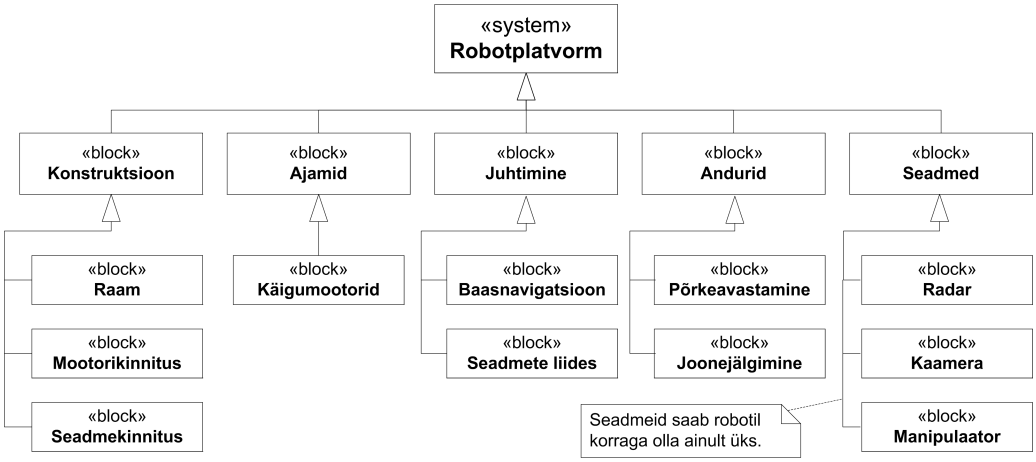


Abbildung 7.1: Model der Systemstruktur

Designlösungen

Für diese Aufgabe hat das Team eine Brainstorming-Methode verwendet und drei vom Konzept völlig unterschiedliche Vorschläge erstellt. Es wurde eine Evaluationsmatrix erstellt und so die optimale Konstruktion gefunden. Der größte Unterschied der drei Lösungen liegt im Bewegungsschema.

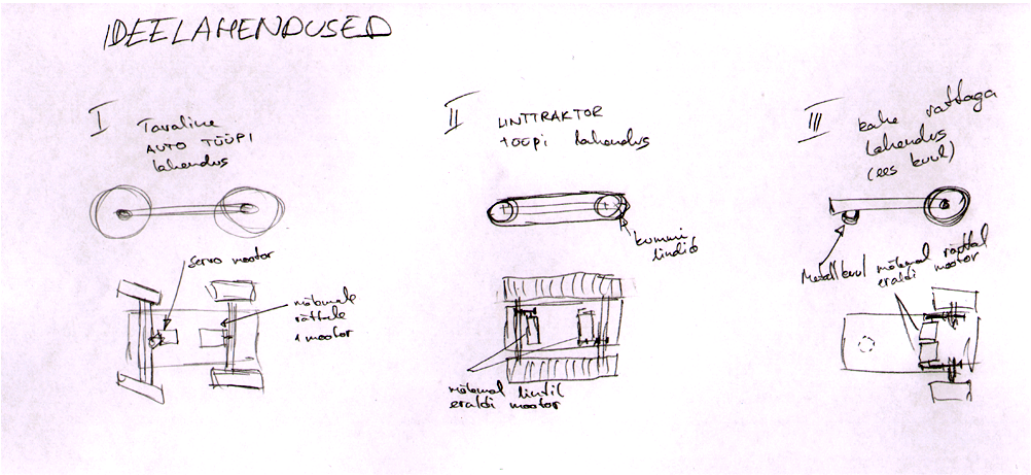


Abbildung 7.2: Designlösungen

Vereinfachte Evaluationsmatrix:

Funktion/Lösung	I	II	III	Gewichtungs-faktor
Kosten	3	4	6	0,8
Komplexität im Bau	2	4	7	0,7
Beweglichkeit	4	8	8	0,5
Permittivität	5	8	2	0,3
Anwendbarkeit in Homelab	5	4	5	0,9
Gewicht	5	6	7	0,8
Total (mit Gewichtungsfaktor)	**19**	**27**	**28**	

Die Evaluationsskala umfasst Werte von 1 bis 10 und der Gewichtungsfaktor liegt zwischen 0 und 1. Der Gewichtungsfaktor wird nach den Anforderungen und Einschränkungen des Systems ausgewählt. So ist beispielsweise Lösung 2 beweglicher auf unebenen Boden, dieses war jedoch nicht in der ursprünglichen Aufgabe gefordert, daher ist der Gewichtungsfaktor niedrig.

Basierend auf der Auswertung, ist die optimale Lösung für die gestellte Aufgabe eine Plattform auf zwei Rädern mit zwei separaten Motoren. Die weitere Arbeit besteht in der Entwicklung der gewählten Lösung in ein reales System.

Mechanik

Die Mechanik wurde so einfach wie möglich gehalten, wobei dem Prinzip der Modularität gefolgt wurde. Front- und Heckstossstange sind identische Module. Die Elektronik beseteht aus drei Modulen, welche übereinander platziert werden und mit einfachen Flachbandkabeln verbunden werden. Darüber hinaus wird so sicher gestellt, dass die Module relativ einfach ausgewechselt werden können.Die Motoren entstammen dem HomeLab kit: Motoren mit einem integrierten Drehzahlminderer und Coder, welche direkt mit den Antrieb des Motors verbunden sind. Es wurden Modellflugzeugräder benutzt, da diese sehr leicht und robust sind.

Um die Konstruktion zu vereinfachen sind Boden und Dachplatte identisch. Die Platten haben Löcher, damit unterschiedliche Geräte an der Dachplatte angeschlossen werden können. Neben der Elektronik passt auch eine Batterie problemlos zwischen den Platten.

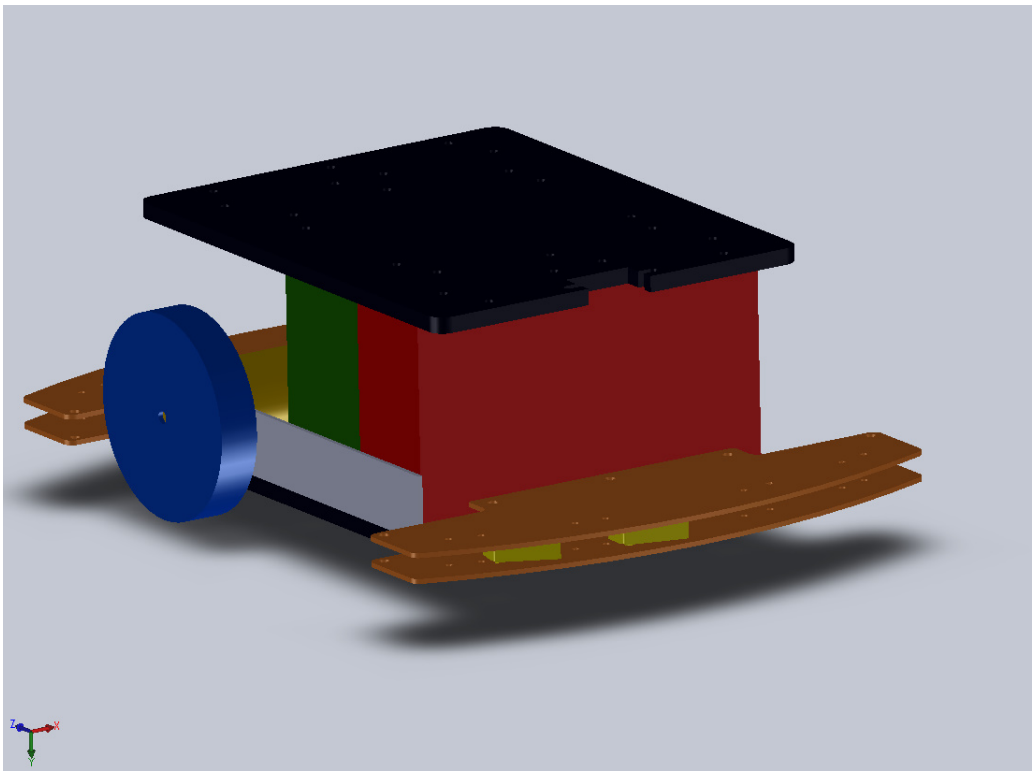


Abbildung 7.3: Erstes 3D-Modell des Roboters und Anordnung seiner Komponenten.

Die Stoßstangen des Roboters werden separat geplant und sind mit Berührungs- und Linienfolgesensoren ausgestattet. Die Stoßstange wird aus PCBs hergestellt und verfügt dadurch über Elektrizität. Die Linienfolgesensoren werden direkt auf die Stoßstange der Bodenplatte gelötet. Berührungssensoren (Mikroschalter) werden zwischen den Stoßstangenplatten platziert und mit einem Gummistück an der Front geschützt. Das Gummistück absorbiert den Aufprall und ermöglicht gleichzeitig zu identifizieren, woher der Aufprall kam.

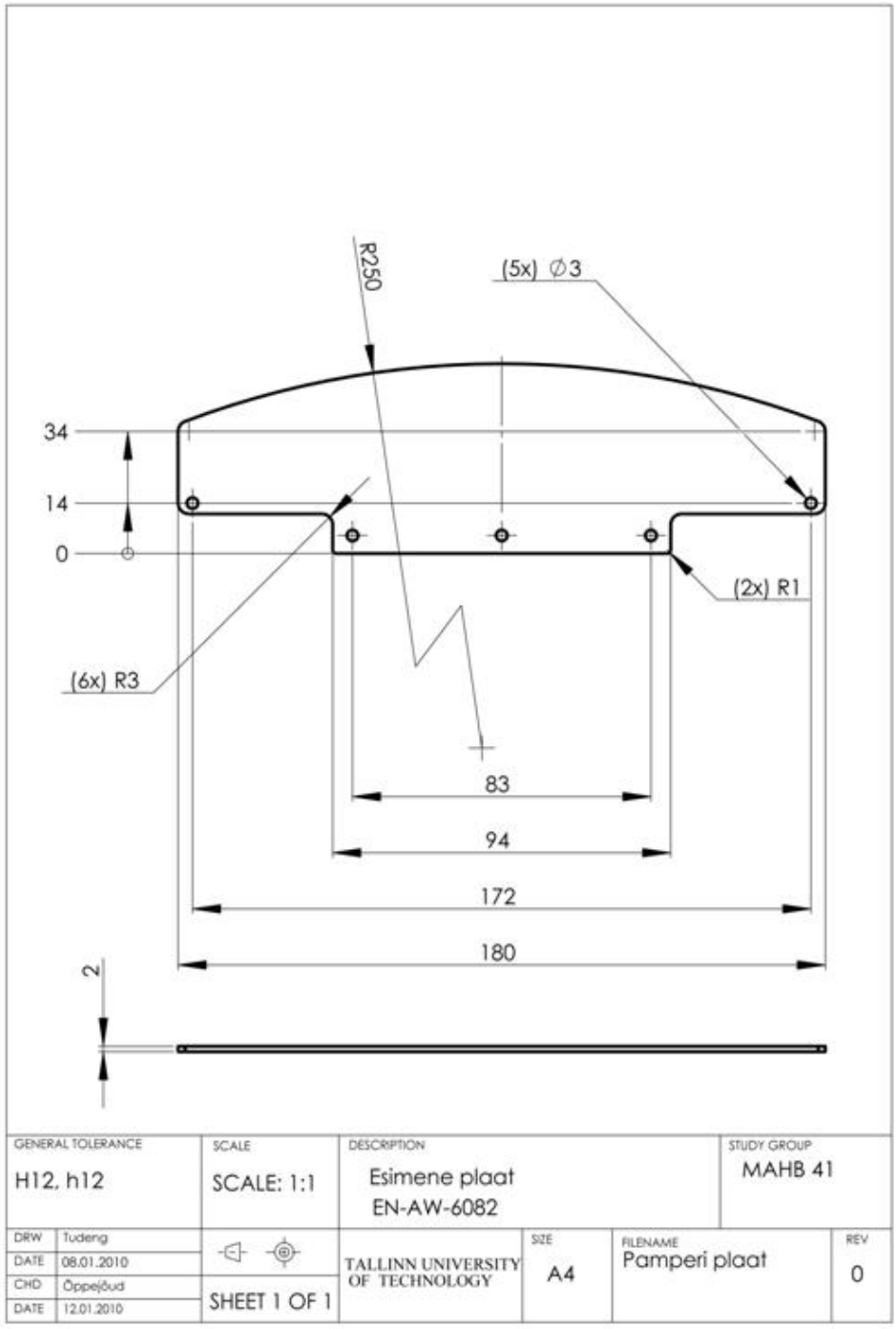


Abbildung 7.4: Zeichnung der Stoßstangenplatte

Elektronik

Die Elektronik des Systems ist in einem Prinzipsschema und Schaltplan mit PCB-Aufbau-Plan beschrieben.

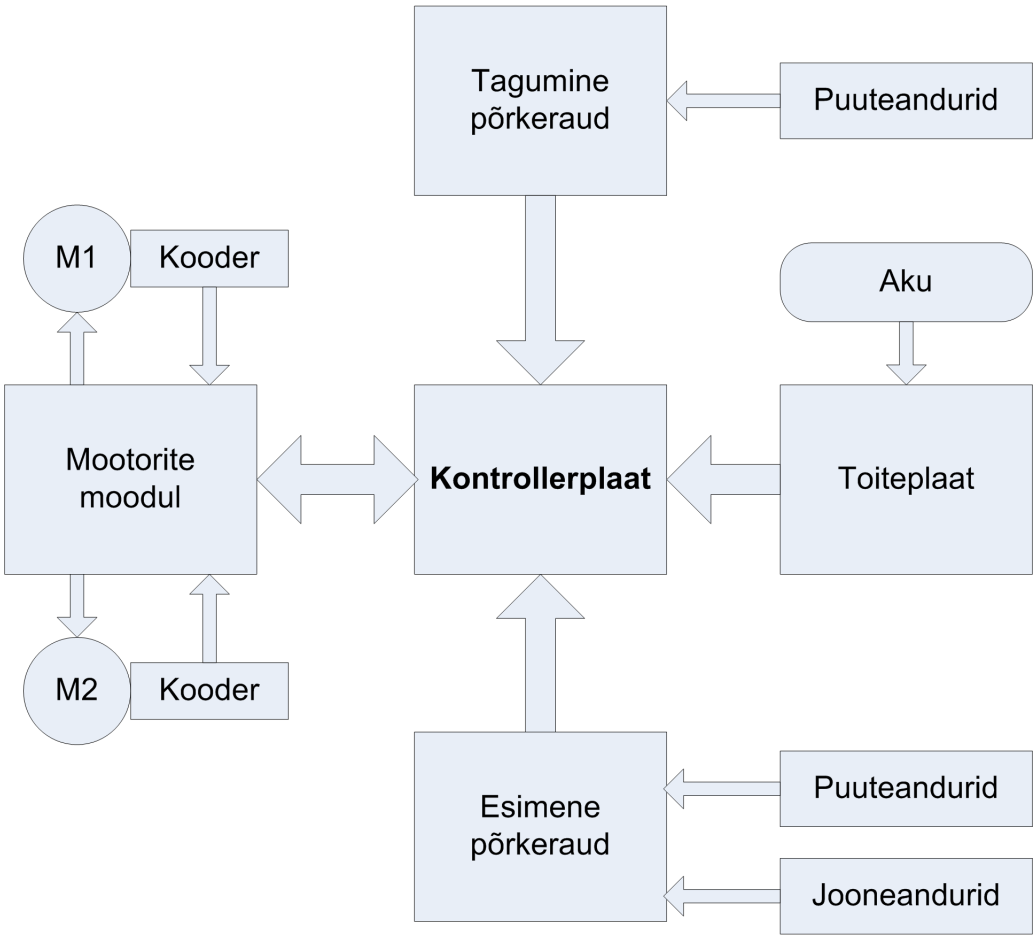


Abbildung 7.5: Blockdiagramm der elektronischen Komponenten

Als Beispiel werden die Schaltpläne der Linienfolgesensoren sowie die dazugehörigen PCB-Aufbau Pläne der Stoßstange des Roboters gezeigt.

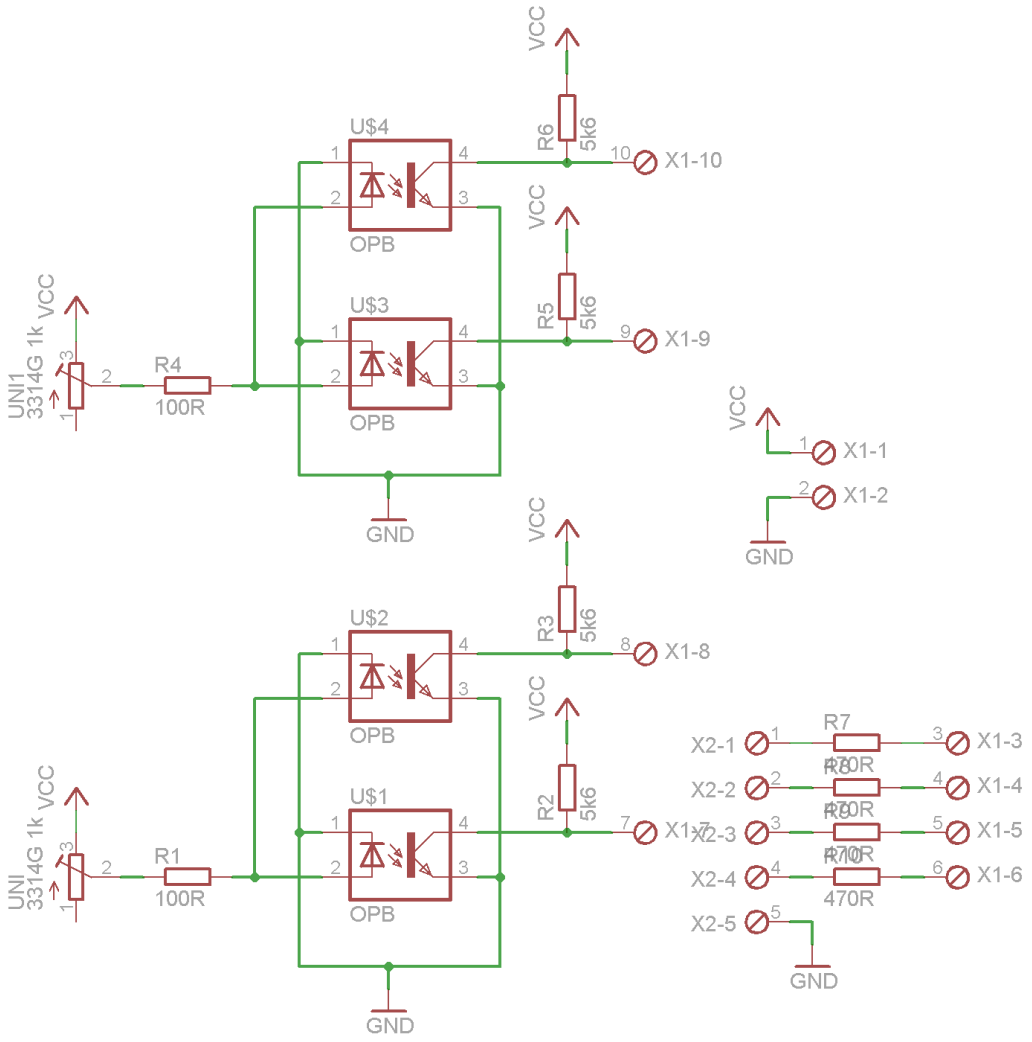


Abbildung 7.6: Schaltplan der Stoßstangensensoren.

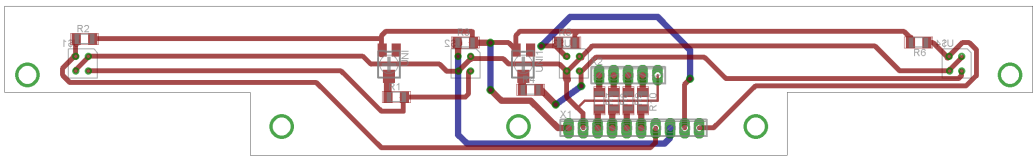


Abbildung 7.7: Aufbauplan der Stoßstange.

Kontrollsystem

Das Kontrollsystem des Roboters, leitet sich von einem Verhaltensmodell ab und ist an Funktionalität, Anforderungen und Einschränkungen der ursprünglichen Aufgabe angepasst.Aus dem Verhaltensmodell des Systems wird ein spezifisches Kontrollprogramm entworfen, welches wiederum die Basis für den Softwareprogrammcode ist.Alle drei Ebenen (Verhaltensmodell-Algorihtmus-Quellcode) müssen miteinander vereinbar sein.

Algorithmus Der Algorithmus beschreibt die Kontrolllogik des System und wird als Blockdiagramm dargestellt. Einige Elemente und Beschreibungen der Verhältnisse genügen, um einen einfachen Algorithmus zu erstellen. Wenn der Algorithmus für den Roboter richtig zusammengestellt ist, kann daraus sehr einfach ein Kontrollprogramm erstellt werden.Grundsätzlich werden zwei verschiedene Objekte in einem Algorithmus genutzt: ein Rechteck mit runden Ecken, zur Darstellung einer Aktivität sowie ein kleiner Diamant zur Kontrolle einer Bedingung, gefolgt von einem Start weiterer Aktivitäten, je nach Resultat der Kontrolle.

Bedeutung der im Algorithmus verwendeten Symbole:

Symbol	**Bedeutung**	**0**	**1**	**-1**
M1	linker Motor	Stop	rotiert im Uhrzeigersinn	rotiert gegen den Uhrzeigersinn
M2	rechter Motor	stop	rotiert im Uhrzeigersinn	rotiert gegen den Uhrzeigersinn
F	erster mittlerer Berührungssensor	kein Signal	Signal	
FR	erster rechter Berührungssensor	kein signal	Signal	
FL	erster linker Berührungssensor	kein signal	Signal	
d	Bezug			

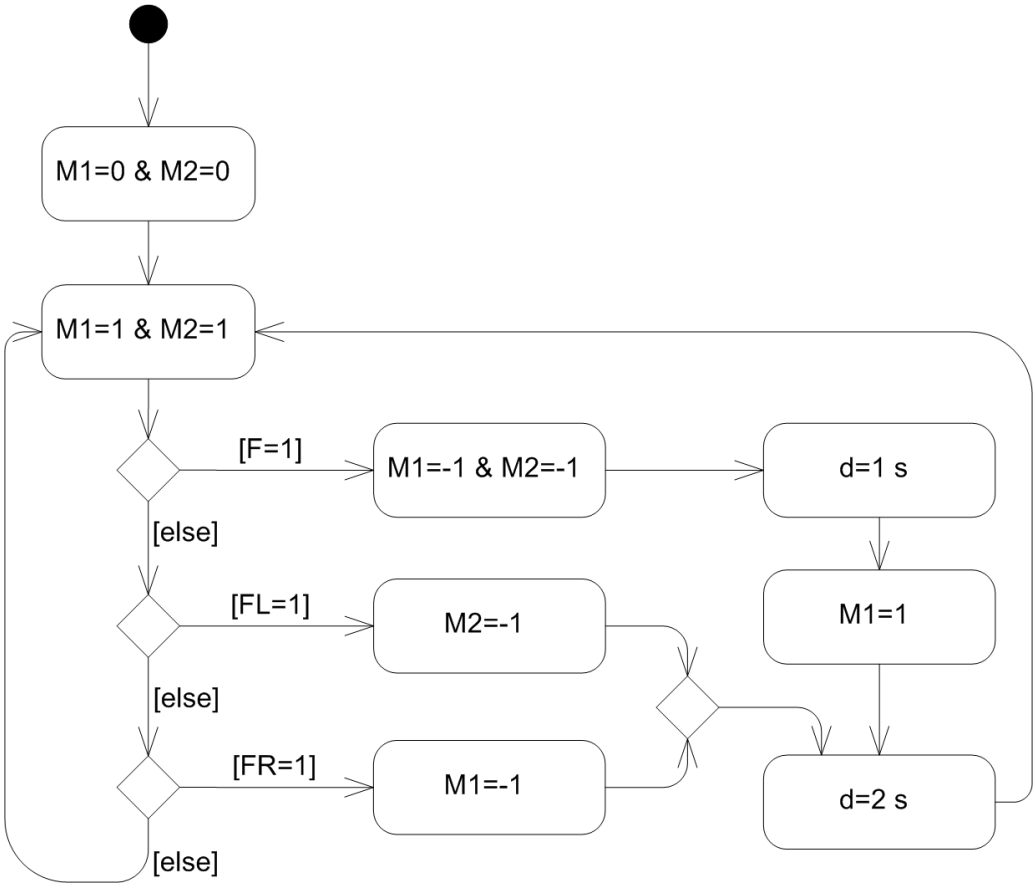


Abbildung 7.8: Statusdiagramm eines Algorithmus

Quellcode Einfache Navigation

```
#include <homelab/module/motors.h>
#include <homelab/pin.h>
#include <homelab/delay.h>

// Definieren der Stoß stangenpins
pin front      = PIN(C, 0);
pin frontleft  = PIN(C, 1);
pin frontright = PIN(C, 2);

//
// Hauptprogramm
//
int main(void)
```

```
{
  // Starten von Motor 0 und 1
  dcmotor_init(0);
  dcmotor_init(1);

  // Sensorpins als Inputs
  pin_setup_input_with_pullup(front);
  pin_setup_input_with_pullup(frontleft);
  pin_setup_input_with_pullup(frontright);

  // Endlosschleife
  while (true)
  {
    // Motorstart im Uhrzeigersinn
    dcmotor_drive(0, 1);
    dcmotor_drive(1, 1);

    // Kontrolle des mittleren Sensorsignals
    if (pin_get_value(front))
    {
      // Umkehr der Motoren
      dcmotor_drive(0, -1);
      dcmotor_drive(1, -1);

      // Pause 1 Sekunde
      sw_delay_ms(1000);

      // Start des linken Motors im Uhrzeigersinn
      dcmotor_drive(0, 1);

      // Pause 2 Sekunden
      sw_delay_ms(2000);
    }

    // Kontrolle des linken Sensorsignals
    else if (pin_get_value(frontleft))
    {
      // Umkehr des rechten Motors
      dcmotor_drive(1, -1);

      // Pause 2 Sekunden
      sw_delay_ms(2000);
    }

    // Kontrolle des rechten Sensorsignals
```

```
    else if (pin_get_value(frontright))
    {
      // Umkehr des linken Motors
      dcmotor_drive(0, -1);

      // Pause 2 Sekunden
      sw_delay_ms(2000);
    }
  }
}
```

Gebrauchsfertige Lösung

Die in diesem Projekt gebaute Roboterplattform besteht zum größten Teil aus Plastik, außer der Motorbefestigung, welche aus Aluminium hergestellt ist. Die elektrischen Module wurden auf einander platziert, die Batterie liegt lose zwischen den Platten. Die Stoßstangen wurden aus PCB gebaut und schwarz gestrichen. Die Dachplatte des Roboters ist flach und erlaubt das Anbringen verschiedener Geräten. Es wurde ein einfaches Radar installiert, welches aus einem kleinen RC Servomotor und einem Infrarotsensor besteht.Als zweite Lösung wurde ein intelligentes Kameramodul installiert, welches beim Lösen visueller Probleme hilft. Beide Lösungen werden auf den folgenden Bildern vorgestellt. Ein einfacher Manipulator wurde als drittes Gerät getestet, dessen Komponenten mit standard Servomotoren sowie einer seriellen Schnittstelle zur Kontrolle des Antriebs betrieben werden.

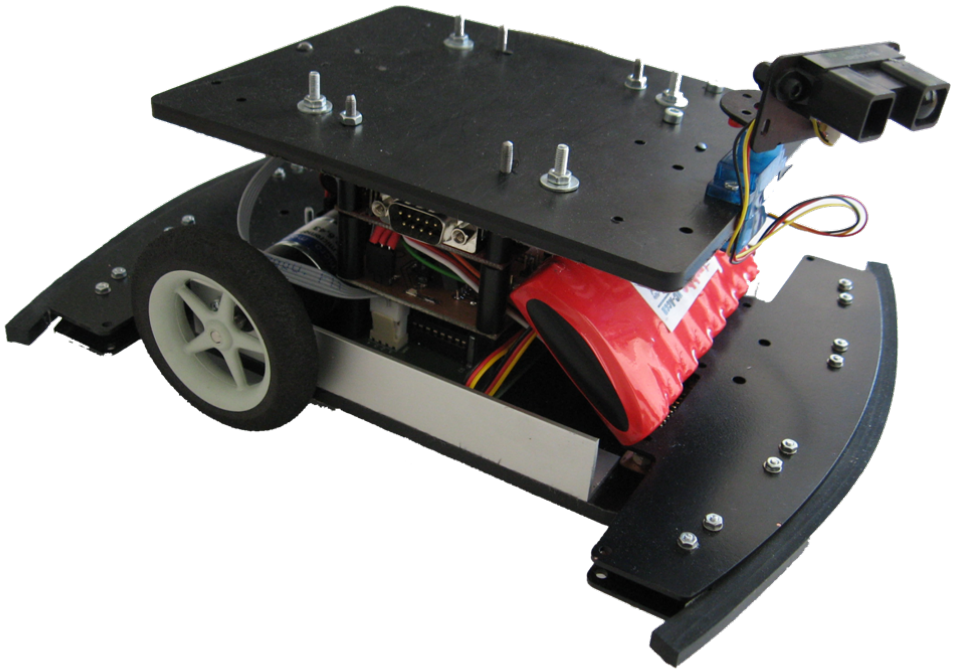


Abbildung 7.9: Roboter mit Infrarotradar.

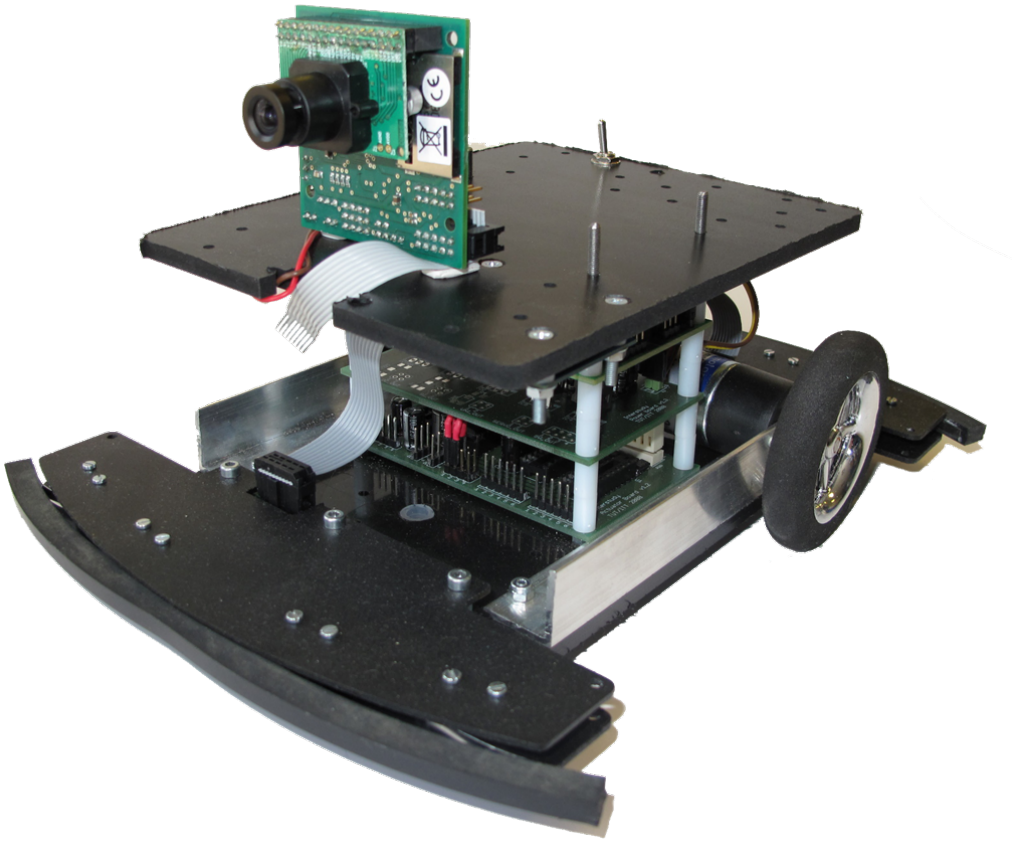


Abbildung 7.10: Roboter mit intelligentem Kameramodul (CMUcam3).

Kostenkalkulation

Die Kostenkalkulation enthält die Kosten für die Komponenten und die Produktion des Roboters

Tabelle der Komponentenkosten

Komponent	Marke	Anzahl	Preis	Kosten
Motor	M LE149.6.43	2	500.-	1000.-
Mikrocontroller	uC ATmega128	1	900.-	900.-
Motorenantriebsplatine	Actuator Board v1.2	1	700.-	700.-
Power plate	TP	1	500.-	500.-
Linienfolgesensoren	LFS QRD1114	8	30.-	240.-
Berührungssensoren	TS Microswitch	8	25.-	200.-
Gehäuseblech	ABS	4	50.-	200.-
PCB Rohteil		2	50.-	100.-
Motorenbefestigungsprofil	Al-L	2	10.-	20.-
Reifen	60/10 mm	2	30.-	60.-
Batterie	NI-MH 9,6 V	1	350.-	350.-
Kabel		10	20.-	200.-
Muttern		1	50.-	50.-
Weiteres Zubehör		1	100.-	100.-
Total				**4620.-**

Kosten in EEK.

Geschätzte Arbeits- und Produktionskosten für ein einzelnes Modell.

Arbeit	Zeit (h)	Preis	Kosten
Fräsen der Bauteile	1	300.-	300.-
Fräsen der PCBs (Stoßstangen)	0,5	500.-	250.-
Bau des Roboters	0,5	250.-	125.-
Bau der Stoßstangen (löten der Komponenten)	1	300.-	300.-
Programmierarbeit	5	300.-	1500.-
Erstellen der Dokumentation	3	250.-	750.-
Total	**11**		**3225.-**

Vorraussichtliche Kosten des Roboters **7845.-**

Die Kostenkalkulation des Roboters basiert auf geschätzten, da es sich um ein Lehrprojekt handelt und aus diesem Grund mehr Zeit für die Arbeit und Konstruktion verwendet wird und keine direkte Bezahlung erfolgt. Daher spiegelt der berechnete Arbeitsaufwand keine reale Situationen wider.

Projektmanagement

Das mechatronische System (Roboter) ist in Gruppenarbeit mit straffem Zeitplan und Budget entstanden, und besitzt damit die wichtigsten Eigenschaften eines Projektes. Die Schlüsselaktivitäten des Projekts waren: Erstellung des Zeitplans, Planung und Management der Gruppenarbeit, Überwachung des Budgets sowie Besorgung des Materials, Weitergabe aktueller Berichte an den Leiter, Präsentation sowie Dokumentation des Ergebnisses. Der Projekbericht enthält Angaben zu den Arbeitsgruppen, Terminen von Meetings, dem Projektplan (am besten in einem Gantt Diagramm), der Ressourcenverteilung (inkl. Personalverteilung) sowie zum geplanten und aktuellen Budget. Nachfolgend wird ein einfacher Projektplan als Gantt Diagramm dargestellt.

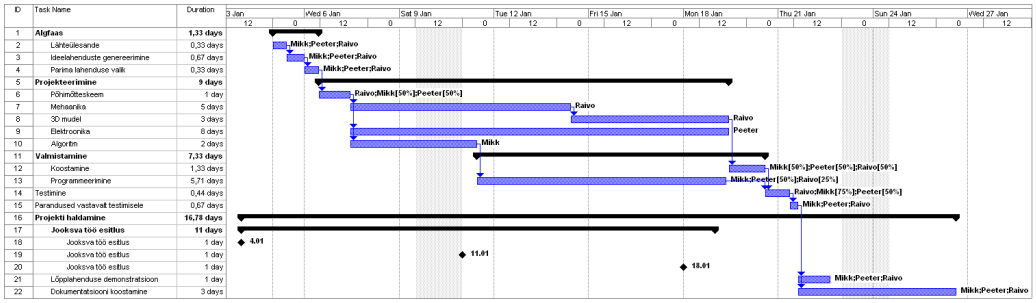


Abbildung 7.11: Projektplan

Fazit

Die Kostenrechnung hat gezeigt, dass die Produktionkosten des Roboters sehr hoch sind, besonders wenn man nur mit einem Exemplar rechnet. Die Kosten blieben jedoch im anfänglich gesetzten Rahmen. Die Prouktionskosten könnten durch Optimierung des Materials und der Komponeten sowie durch gleichzeitige Fertigung mehrerer Roboter wesentlich reduziert werden.Während des Projekts ist deutlich geworden, wie ein mechatronisches System geplant, gebaut und getestet wird.

Am Ende der Arbeit ist eines klar geworden: Damit der Roboter richtig funktioniert, sollte deutlich mehr Zeit zum Testen eingeplant werden, insbesondere für Softwaretests.Unterschiedliche Module funktionieren nicht immer einwandfrei zusammen, auch wenn dieses in einzelnen Experimenten klappt. Dadurch wird deutlich, dass die Integration von Modulen in ein System eine echte Herausforderung darstellt, und hierfür mehr Zeit und Ressourcen eingeplant werden sollten.

Abschließend hoffen wir, dass das Projekt sehr interessant und lehrreich war und einen Einblick in das Design und die Konstruktion von integrierten Systemen geben konnte.

Genutzte Quellen und Materialien

1. Allgemeines Benutzerhandbuch des HomeLab http://home.roboticlab.eu

2. Datenblatt des ATmega128

3. Dudziak, R., Köhn, C., Sell, R., Integrated Systems & Design, TUT Press, 2008

4. Friendenthal, S., Moore, A., Steiner, A., A Practical Guide to SysML, Elsevier, 2008

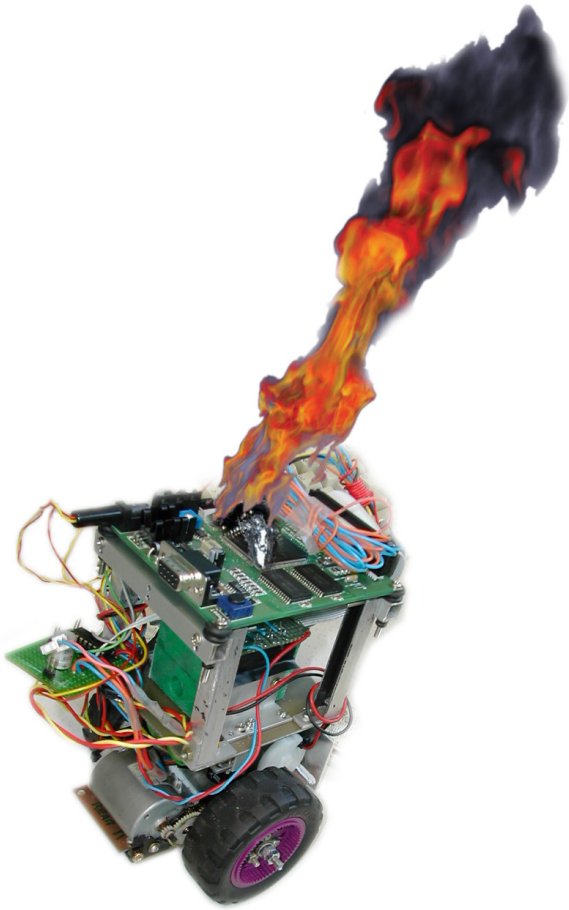
5. Perens, A. Project Management, Külim, 1999

6. Bräunl, T. Embedded Robotics, Springer-Verlag, 2003

8 Ausblick

Wenn Sie am Ende dieses Buches angelangt sind, haben Sie einige Kenntnisse sowie praktische Fähigkeiten in Bezug auf das Programmieren von Mikrocontrollern und den Umgang mit diversen Geräten erlangt, selbst wenn Sie nicht alle Übungsbeispiele absolviert haben. Wir hoffen, dass dieses Buch Sie bei der Bewältigung von Problemen sowie der Erlangung theoretischen Hintergrundwissens unterstützen konnte. Der nächste Schritt wäre nun, eigene intelligente Geräte zu entwickeln, die Ihren Interessen oder Bedürfnissen angepasst sind. Schüler und Studenten können sich darüber hinaus nach Robotikwettbewerben umsehen. Auf jeden Fall können Sie das erworbene Wissen dazu nutzen, Ihren Alltag einfacher und mit mehr Freude zu gestalten. Sie können einfache Sicherheitssysteme für Ihr Ferienhaus oder Ihre Garage entwickeln, die mit verschiedenen Sensoren arbeiten, um Einbrecher aufzuspüren und den Eigentümer zu warnen. Bei Problemen, fragen Sie um Hilfe. Dieses Buch wird sich ständig in seiner elektronischen Form weiter entwickeln. Es ist Teil eines Robotik-Studienkonzepts, welches die Pflege der Webseite einschließt. Zusätzlich zum online verfügbaren Material existiert ein Benutzerforum, in welchem sie Antworten auf Ihre Probleme finden können. Die elektronische Version enthält zusätzliche Kapitel zu den Themen Bluetooth, Ethernet, RFID, Machine Vision für Fortgeschrittene etc.

Für Lehrer: Registrieren Sie sich auf unserer Homepage und teilen Sie uns kurz mit, dass Sie Lehrer sind. Daraufhin werden Sie Zugriff auf die Lösungen der Übungsaufgaben sowie Antworten auf die Fragen am Ende jedes Übungsabschnittes erhalten.



Homepage:

http://www.roboticlab.eu